AMSTRAD

CREER DE NOUVELLES INSTRUCTIONS

JEAN-CLAUDE DESPOINE



AMSTRAD CPC 464
CRÉER DE NOUVELLES INSTRUCTIONS

J.-C. DESPOINE

AMSTRAD CPC 464
CRÉER DE NOUVELLES INSTRUCTIONS



Paris • Berkeley • Düsseldorf

SOMMAIRE

AVANT-PROPOS

L'Amstrad CPC est équipé d'un processeur Z 80, fonctionnant à 4 mégahertz et disposant d'un jeu d'instructions extrêmement puissant. En outre, la qualité de son BASIC, tant du point de vue de la vitesse que du nombre d'instructions, range cette machine parmi les plus performantes du marché.

Il n'en reste pas moins que le BASIC sera toujours le BASIC, avec ses inconvénients et ses limites. Qui, par exemple, n'a pas éprouvé d'amères déceptions en voyant un mobile se traîner sur l'écran, avec force soubresauts, au terme d'un programme d'animation en BASIC. De plus, on ne peut que regretter l'absence, sur l'Amstrad, d'instructions telles que PAINT, CIRCLE, etc.

Le langage machine peut remédier à ces défauts.

Lorsque vous travaillez en BASIC sur votre ordinateur, c'est un peu comme si vous teniez une conversation avec, disons, un Anglais, par traducteur interposé. En effet, le processeur n'a pas la moindre idée de ce que signifient des choses telles que PRINT, LOCATE ou GOTO. Et pourtant, ça marche ! Alors ?

Alors il y a effectivement, quelque part dans la machine, un programme intégré qui traduit les instructions BASIC en langage machine, le seul compris par le Z 80.

On voit donc tout de suite l'intérêt qu'il peut y avoir à travailler directement en langage machine : pour reprendre l'exemple cité plus haut, votre conversation sera bien plus efficace et rapide si vous vous exprimez directement en anglais plutôt que de passer par un interprète.

A titre d'exemple, voici deux versions, l'une en BASIC et l'autre en langage machine, d'un même programme qui colore l'écran.
Ne cherchez pas à comprendre ce programme pour l'instant.

Programme BASIC

```
10 X=255
20 MODE I : FOR I=49152 TO 65535
30 POKE I,X : NEXT
```

Vous pouvez faire varier la couleur en affectant à X, en ligne 10, une valeur quelconque de 0 à 255.

Programme en langage machine

```
10 MODE 1 : MEMORY 40000 : X=255
20 FOR I=40001 TO 40016 : READ a : POKE I,a : NEXT
30 POKE 40008,X : CALL 40001
40 DATA &21, &0, &40, &11, &FF, &BF, &3E, &0, &13,
   &12, &2B, &7C, &B5, &20, &F7, &C9
```

Là encore, vous pouvez faire varier la valeur de X entre 0 et 255. Pour la même tâche, on peut constater que le programme BASIC met environ 36 secondes, alors que le même en langage machine met un peu plus de deux dixièmes de seconde !

Cette vitesse d'exécution va naturellement nous permettre des réalisations qui seraient impossibles en BASIC.

Il n'est bien entendu pas question pour nous de remplacer totalement le BASIC et d'écrire de gigantesques programmes en langage machine, mais simplement d'apprendre à construire des petites routines (ou sous-programmes) que nous utiliserons comme s'il s'agissait d'instructions nouvelles.

S'il est vrai que le langage machine est beaucoup moins tolérant que le BASIC en ce qui concerne les erreurs, et en dépit de son aspect quelque peu rébarbatif pour le profane, vous vous rendez vite compte que le premier n'est pas beaucoup plus difficile à travailler que le second.

Par ailleurs, et contrairement à ce que pourraient laisser croire un certain nombre d'ouvrages s'étendant avec complaisance sur le sujet, aucune connaissance particulière de l'arithmétique binaire, ou hexadécimale n'est nécessaire (à moins, bien sûr, de vouloir devenir un spécialiste).

Le BASIC de l'Amstrad nous offre en effet deux précieuses fonctions capables d'effectuer pour nous toutes les conversions nécessaires : il s'agit des fonctions HEX$ et BIN$, dont nous reparlerons.

Tout au plus, concernant l'hexadécimal, vous faudra-t-il apprendre à distinguer un octet fort d'un octet faible.

Ce problème est abordé dans la Première Partie, consacrée aux connaissances de base minimales à acquérir avant d'aborder la programmation proprement dite.

I
CONNAISSANCES DE BASE

LA NOTATION HEXADÉCIMALE : OCTETS FORTS ET OCTETS FAIBLES

Il serait toujours possible, si on le souhaitait vraiment, de travailler en langage machine en fournissant à l'ordinateur des données sous une forme que tout le monde connaît : la notation décimale.

Pour un certain nombre de raisons, cela n'est pourtant pas recommandé. On peut en citer trois :

• Certaines conversions en hexadécimal seraient de toute façon nécessaires lors de la conception du programme.

• La numérotation en hexadécimal prend moins de place que celle en décimal (pour les nombres supérieurs à 99 en particulier).

• Tous les ouvrages sans exception ayant trait à ce sujet ou à un sujet approchant se servent de la notation hexadécimale. Autant donc s'y habituer tout de suite...

Pour découvrir de quoi il s'agit exactement, commencez par taper ceci sur votre machine (le mode direct suffira, c'est-à-dire qu'il est inutile de mettre un numéro de ligne) :

```
PRINT HEX$ (43870)
```

La réponse qui s'affiche est : AB5E. La fonction HEX$ a transformé la représentation décimale de 43870 en sa représentation hexadécimale. Les deux chiffres ou lettres de droite sont appelés l'*octet de poids faible,* et les deux de gauche l'*octet de poids fort* (dans l'exemple, l'octet de poids faible est 5E et l'octet de poids fort AB).

Il faut noter que l'ordinateur ne prend pas la peine d'écrire les 0 éventuellement situés à gauche du dernier chiffre ou de la dernière lettre.

Exemples

2060 est représenté en hexadécimal par 80C, qui est l'équivalent de 080C. L'octet de poids faible est donc 0C et l'octet de poids fort 8.

HEX$ (255) = FF, ce qui est l'équivalent de 00FF. Dans ce cas, l'octet de poids faible est FF et l'octet de poids fort 0 (ou, si l'on veut, il n'y a pas d'octet fort — nous verrons toutefois que la nuance peut être importante).

La conversion inverse (hexadécimal → décimal) peut s'effectuer de la manière suivante :

Nombre en décimal = octet de poids faible + (256 * octet de poids fort)

Exemple

HEX$ (648) = 288 et &88 + 256 * &2 = 648

Le signe & signale à l'ordinateur que le chiffre qui lui est transmis est en notation hexadécimale.

Il faut enfin signaler que l'on ne peut représenter en hexadécimal que des nombres de – 32768 à 65535.

Les nombres négatifs posent toutefois un problème particulier, que nous laisserons de côté, puisqu'il ne se présentera pas dans nos programmes. Néanmoins, pour ceux que cela intéresse, qu'ils sachent que :

1. La représentation hexadécimale des nombres allant de – 32768 à – 1 est la même que celle des nombres de 32768 à 65535 (exemple : HEX$ (– 21) = FFEB et HEX$ (65515) = FFEB).
2. Pour faire la conversion hexadécimal → décimal, il existe donc en réalité deux possibilités :

- Si l'on ne veut pas tenir compte du signe (si l'on sait par exemple que le nombre est positif), on utilise la formule présentée plus haut.
- Si l'on veut tenir compte du signe, il faut utiliser une fonction qui a pour nom UNT.

Exemple

UNT (&FFEB) = – 21 et &EB + 256 * &FF = 65515

Mais, répétons-le, tout cela n'a pour le moment aucune importance.

LES REGISTRES SIMPLES

Pour en donner une idée claire, on pourrait dire que la programmation en langage machine consiste pour une grande part en une manipulation de différentes "boîtes" dans lesquelles il est possible de mettre des valeurs, et que l'on peut par exemple additionner et soustraire entre

elles. Chacune de ces boîtes, que l'on appelle registre, a un nom. Nous nous intéresserons pour l'instant à sept de ces registres : A, B, C, D, E, H et L.

Pour information, précisons que le registre A s'appelle également l'*accumulateur*.

Ces registres simples, pour utiles qu'ils soient, ont pourtant un grave défaut : on ne peut y mettre que des nombres de 0 à 255. C'est pourquoi il existe également les registres doubles.

LES REGISTRES DOUBLES

Ce sont, en fait, des registres simples mis deux à deux. Nous en utiliserons trois : BC, DE et HL.

Dans ces registres doubles, on pourra mettre des nombres jusqu'à 65535. C'est là que vont intervenir les notions d'octet faible et d'octet fort.

Pour mettre par exemple la valeur &AB5E (43870 en décimal) dans le registre HL, il faudra mettre (on dit aussi charger) l'octet fort de cette valeur dans H et l'octet faible dans L. Une fois chargé, HL se présentera donc comme ceci :

H	L
AB	5E

Autres exemples

&80C (2060) dans BC :

B	C
8	0C

&D2 (210) dans DE :

D	E
0	D2

Ce dernier exemple amène une remarque importante : pour charger un registre double avec un nombre inférieur ou égal à 255 (donc n'ayant pas d'octet fort), il faut *considérer que l'octet fort vaut 0 et le charger dans le registre correspondant.*

Pour conclure, signalons enfin que nous serons amenés à nous servir également d'un registre double un peu particulier, le registre IX. Ce registre est appelé le *registre d'index.*

LA PILE

Pour nous, la pile ne sera rien de plus que ce que son nom indique : un certain nombre de données empilées les unes sur les autres. Peu importe de savoir comment elle est gérée par la machine, l'essentiel étant de savoir l'utiliser.

Traditionnellement, la comparaison utilisée pour parler de la pile est celle de la pile d'assiettes, et il faut reconnaître qu'il est difficile de trouver mieux.

Imaginons que vous soyez en train de faire la vaisselle. Vous lavez d'abord une assiette rouge que vous posez quelque part en attendant le rinçage. Vous en lavez ensuite une bleue que vous posez sur la rouge, puis une jaune que vous posez sur la bleue. Le lavage terminé, vous attaquez maintenant le rinçage. Si vous prenez vos assiettes comme elles se présentent sur la pile, la première à être rincée sera la dernière qui a été posée, en l'occurrence la jaune.

C'est le principe de base de la pile : dernier élément entré, premier élément sorti (en anglais, c'est une pile LIFO *Last In/First Out).*

Si, par superstition ou par fantaisie, vous aviez absolument voulu rincer l'assiette rouge d'abord, il vous aurait au préalable fallu soulever la jaune et la bleue pour prendre la rouge.

Dans nos programmes, nous utiliserons souvent la pile pour y stocker provisoirement des données, car c'est un moyen commode et rapide. Son fonctionnement peut paraître simple, pour ne pas dire simplet, mais vous verrez que l'on a vite fait de s'y perdre, si l'on n'y prend pas garde, lorsque les données s'accumulent.

De plus, il faut savoir (et ne jamais l'oublier) que le processeur, de son côté, se sert lui aussi de la pile pour faire son travail, et qu'il ne doit pas y avoir d'interférence entre vous et lui. Nous en reparlerons...

LE REGISTRE D'INDICATEURS

C'est un registre que nous n'utiliserons pas directement et, là encore inutile de s'encombrer de détails superflus. Il suffit de nous imaginer ce registre comme une boîte comportant huit cases, ou indicateurs, ayant chacune un numéro, et parfois un nom :

7	6	5	4	3	2	1	0
S	Z	—	H	—	P/V	N	C

Dans chacune de ces "cases", il peut y avoir 1 ou 0 : on dit que tel indicateur est mis à 1 ou 0.

Nous ne toucherons jamais au contenu de ce registre, c'est l'affaire personnelle du processeur. Nous ne nous intéresserons qu'aux indicateurs 7 ou 6 dont voici le rôle :

Tout au long du déroulement d'un programme, et après certaines opérations, le processeur met les indicateurs 6 et 7 à 1 ou à 0, en fonction du résultat.

- Si le résultat de l'opération est nul, l'indicateur 6 sera mis à 1. Sinon, il sera mis à 0. (Le mot opération doit être pris ici au sens large. Une comparaison entre valeurs est par exemple une opération au même titre qu'une addition.)
- Si le résultat de l'opération est positif, l'indicateur 7 sera mis à 1. S'il est négatif, l'indicateur 7 sera mis à 0.

L'indicateur 6 est appelé l'indicateur de zéro (Z), et le 7 l'indicateur de signe (S).

Encore une fois, tout cela est géré par la machine et nous n'aurons pas à nous en préoccuper. L'intérêt est néanmoins évident : comme il existe des instructions permettant de tester les indicateurs, nous pourrons effectuer des branchements conditionnels du genre : "Si l'indicateur 7 est à 1, alors il faut faire ceci ou cela..."

C'est en fait exactement comparable à l'instruction IF du BASIC.

Remarque

Certaines opérations n'entraînent aucune modification des indicateurs, et cela quel que soit le résultat. En outre, certaines instructions conditionnelles ne peuvent s'utiliser qu'en fonction par exemple de l'indicateur 6, et non pas en fonction du 7 (ou vice versa).

Tout cela est précisé dans la cinquième partie qui résume et explique d'une manière formelle une partie des instructions du Z 80.

LA MÉMOIRE

Le CPC est un ordinateur ayant une RAM (mémoire vive) de 64K, ce qui signifie 64×1 024 = 65 536 octets ou cases mémoire, ayant chacune une adresse de 0 à 65535.

En réalité, une partie de ces cases mémoire est réservée à des choses comme la gestion de l'écran ou l'interpréteur BASIC, et il ne reste à l'utilisateur que 43 536 octets (de l'adresse 368 à 43903), c'est-à-dire pas tout à fait 43K.

Lorsque vous écrivez un programme BASIC, il est codé par la machine et stocké à partir de l'adresse la plus basse (368). Plus le programme s'allonge, plus la mémoire se remplit. En outre, il peut arriver que l'exécution d'un programme nécessite des cases mémoire dans les adresses les plus hautes (43903 et au-dessous).

Le problème, c'est qu'il nous faudra disposer, pour nos programmes en langage machine, d'une zone protégée où nous pourrons les loger, et où ils ne courront pas le risque d'être dérangés ou "écrasés" par le BASIC. Nous allons nous servir pour cela de l'instruction MEMORY, qui permet de fixer librement une limite que le BASIC ne pourra en aucun cas franchir. Si par exemple nous tapons :

```
MEMORY 10000
```

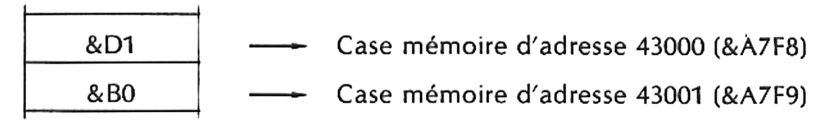
les cases mémoire allant de 10001 à 43903 resteront vierges de toute intrusion, et le BASIC restera confiné entre les adresses 368 et 10000. Il ne s'agit là que d'un exemple, et nous n'aurons fort heureusement pas à réserver des zones de mémoire aussi importantes. Un programme en langage machine de 500 octets, par exemple, représente déjà un programme tout à fait considérable. Ceux que nous vous proposerons dans cet ouvrage tournent autour de 100 ou 200 octets. Un MEMORY 43600 serait donc tout à fait suffisant, la perte de mémoire négligeable, et rapidement amortie.

Nous utiliserons l'espace mémoire ainsi ménagé de deux manières : d'une part pour y écrire nos instructions, et d'autre part, bien que plus rarement, pour y stocker des données.

A cet égard, une case mémoire fonctionne un peu comme un registre, puisqu'on ne peut y mettre que des nombres inférieurs ou égaux à 255. Pour y mettre des nombres supérieurs, il faudra utiliser deux cases mémoire et mettre dans l'une l'octet faible et dans l'autre l'octet fort.

Il importe dès à présent de bien distinguer l'*adresse* d'une case mémoire de son *contenu*.

Exemple



La case mémoire d'adresse 43000 contient la valeur &D1, et celle d'adresse 43001 contient la valeur &B0.

Traditionnellement, le contenu d'une case mémoire d'adresse n s'écrit (n). On peut donc écrire : (43000) = &D1 et (43001) = &B0.

LES INSTRUCTIONS DU Z 80

Le processeur Z 80 possède un nombre considérable d'instructions de base : 158. Pour ce qui nous concerne, nous nous contenterons d'en utiliser 35, et vous pourrez constater qu'avec ce nombre réduit, il est déjà possible de faire un certain nombre de choses.

Une instruction se présente sous la forme d'un ou plusieurs nombres que nous écrirons sous la forme hexadécimale. Ces nombres sont toujours inférieurs ou égaux à 255.

Deux exemples

&19 signifie : "Additionner les registres HL et DE".

&DD, &2B signifie : "Décrémenter (= enlever 1) le registre IX".

Physiquement, un programme en langage machine se présente donc tout simplement comme une série de nombres, écrits dans des cases mémoire les uns à la suite des autres.

LES ROUTINES DU SYSTÈME D'EXPLOITATION

Le CPC n'est pas un égoïste, et il nous offre la possibilité, ô combien précieuse, d'accéder aux routines de son système d'exploitation. Ces routines, ou sous-programmes, font partie des programmes intégrés qui permettent à l'ordinateur de fonctionner. Chacune d'elles ayant une adresse bien précise, elles peuvent être appelées exactement de la même manière que l'on appelle un sous-programme en BASIC grâce à l'instruction GOSUB.

Cette possibilité nous épargnera un travail considérable, puisque nous allons utiliser abondamment ces routines "prêtes à l'emploi".

Si, par exemple, nous avons besoin de calculer le sinus d'un angle, il suffit d'appeler la routine d'adresse &BD88. Bien entendu, il faut savoir, pour chacune d'entre elles, quelles sont les modalités d'appel (pour reprendre l'exemple précédent, où faut-il mettre la valeur de l'angle avant d'appeler la routine sinus, et où se trouve le résultat lors du retour de cette routine ?).

Tout cela est expliqué au fur et à mesure pour les routines que nous utiliserons, et en Annexe V pour celles que nous n'utiliserons pas, mais qui peuvent vous servir pour vos propres programmes.

II
LE LANGAGE MACHINE

COMMENT IMPLANTER UN PROGRAMME EN LANGAGE MACHINE

Nous allons nous servir, pour illustrer cette section, du programme proposé dans l'avant-propos, et qui colorait l'écran (cette fois, pourtant, nous n'y incluerons pas la possibilité de faire varier X).

```
10 MODE 1 : MEMORY 40000
20 FOR I=40001 TO 40016 : READ A : POKE I,A : NEXT
30 CALL 40001
40 DATA &21, &0, &40, &11, &FF, &BF, &3E, &FF, &13,
        &12, &2B, &7C, &B5, &20, &F7, &C9
```

La ligne 10, outre qu'elle initialise l'ordinateur en mode 1, nous intéresse surtout parce qu'elle protège une zone de mémoire, à partir de l'adresse 40000.

La ligne 20 écrit le programme dans une zone de 16 octets :

1. La boucle démarre en 40001.
2. La première donnée stockée en DATA est écrite dans cette case mémoire.
3. La deuxième boucle commence : la deuxième donnée stockée en DATA est écrite en 40002, et ainsi de suite.

Une fois le programme écrit en mémoire, il ne reste plus qu'à l'appeler grâce à l'instruction CALL, suivie de son adresse (c'est-à-dire l'adresse de la première de ses instructions, en l'occurrence 40001). Deux remarques s'imposent :

- La longueur de la zone à protéger dépend bien évidemment de la longueur du programme à y implanter (dans notre cas, un MEMORY 43887 aurait pu suffire, puisque notre programme ne fait que 16 octets).
- Il n'est nécessaire et suffisant que d'exécuter le programme de chargement une seule fois. Lorsqu'il est chargé, on peut l'appeler grâce à CALL autant de fois que l'on veut.

Une petite astuce nous évitera dorénavant d'avoir à mettre le sigle & devant chacune des données de la ligne de DATA. Il suffit de remplacer la ligne 20 par :

```
20 FOR I=40001 TO 40016 : READ A$ : POKE I,
         VAL ("&H"+A$) : NEXT
```

Un dernier conseil enfin, avant de clore le sujet : vous vous rendrez vite compte que rien ne se produit plus facilement, lorsqu'un programme est assez long, qu'une ou deux erreurs lors de la copie des lignes de DATA. Or, les conséquences d'une erreur dans un programme en langage machine sont parfois lourdes (nous y reviendrons). C'est pourquoi la ligne suivante, insérée juste après la boucle de chargement, permet en principe de signaler ce genre d'erreur :

```
25 W=0 : FOR I=Adresse de départ TO adresse de fin :
   W=W+PEEK(I) : NEXT : IF W< > Verification THEN
   PRINT "Erreur de data" : END
```

Pour l'exemple précédent, adresse de départ=40001, adresse d'arrivée=40016, et vérification=1742.

La valeur de vérification varie bien sûr selon le programme, elle sera donnée pour chacun de ceux que nous étudierons.

LES PARAMÈTRES TRANSMISSIBLES

Si l'instruction CALL ne pouvait être utilisée que sous la forme que nous avons vue précédemment, il est bien évident qu'elle serait d'un emploi très limité.

Imaginons par exemple que nous fabriquions une routine chargée de dessiner des carrés. Nous aimerions certainement pouvoir indiquer que nous voulons un carré dont les côtés auront telle ou telle longueur, et qui sera situé à tel endroit de l'écran.

L'instruction CALL permet de transmettre ce genre d'indications (paramètres) sous la forme suivante :

```
CALL adresse du programme, X, X1, X2, X3,...
```

On peut ainsi transmettre jusqu'à 32 paramètres !

Reprenons l'exemple du carré. L'appel de la routine pourrait se présenter ainsi (ne pas oublier les virgules) :

```
CALL adresse, X, Y, L
```

X et Y seraient les coordonnées du coin supérieur gauche de notre carré, et L la longueur des côtés.

Ainsi donc, avec un seul programme, nous pourrions dessiner n'importe quel carré.

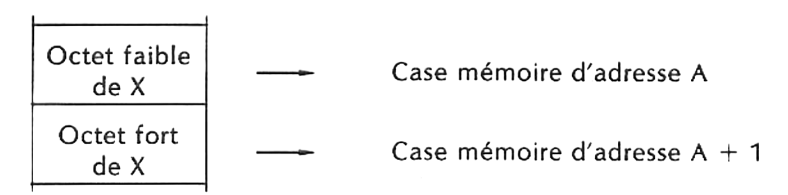
Il ne faut pas être sorcier pour en déduire que nous devrons, pour pouvoir utiliser ces paramètres, être capables de localiser l'endroit où ils sont rangés en mémoire. C'est à cela que nous servira le registre IX.

Considérons d'abord le cas le plus simple, l'instruction CALL suivie d'un seul paramètre :

CALL adresse, X

Précisons tout de suite que l'ordinateur va systématiquement réserver deux cases mémoire pour ranger chaque paramètre, au cas où ils seraient supérieurs à 255.

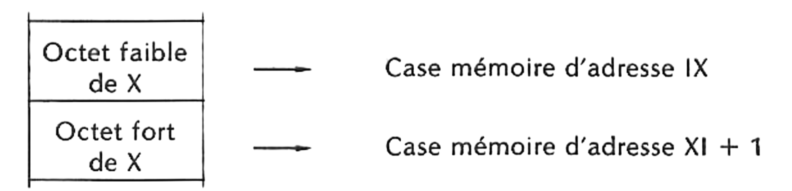
Notre paramètre X se présentera donc quelque part en mémoire sous la forme :



A ce propos, il est important de noter que, lorsqu'un nombre est écrit en mémoire, *l'octet faible est toujours écrit avant le fort.*

Pour en revenir à ce paramètre X, le problème est donc de savoir où il se trouve, en d'autres termes, de trouver son adresse A.

Ce problème est résolu facilement quand on sait que cette adresse est automatiquement chargée dans le registre IX au moment de l'appel du programme (on dit que IX est pointé sur A) :



Cela peut être formulé de différentes manières :

- Octet faible de X = contenu de la case mémoire dont l'adresse est dans IX (on dit "adressée par IX").
- Octet fort de X = contenu de la case mémoire adressée par IX + 1.

Ou encore :

• Octet faible de X = (IX) et octet fort de X = (IX + 1).

Comme il est obligatoire d'utiliser IX avec ce qu'il est convenu d'appeler un déplacement, nous écrirons (IX + 0) au lieu de (IX), ce qui revient exactement au même.

Voyons maintenant un cas plus complexe avec, par exemple, trois paramètres :

CALL adresse, X, X1, X2

Ils seront rangés en mémoire de la manière suivante :

Octet faible de X2	⟶	adresse IX + 0
Octet fort de X2	⟶	adresse IX + 1
Octet faible de X1	⟶	adresse IX + 2
Octet fort de X1	⟶	adresse IX + 3
Octet faible de X	⟶	adresse IX + 4
Octet fort de X	⟶	adresse IX + 5

On remarquera que *le dernier paramètre est rangé le premier* dans la mémoire.

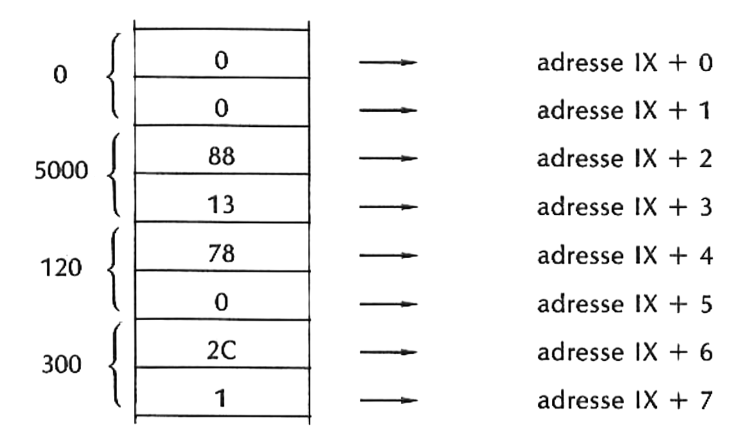
Imaginons maintenant qu'au cours d'un programme nous voulions charger le paramètre X1 dans le registre HL. Il nous suffirait de charger (IX + 3) dans H et (IX + 2) dans L. Bien entendu, il existe des instructions pour cela.

Une bonne compréhension du processus étant importante, voici un exemple concret avec quatre paramètres :

CALL adresse, 300, 120, 5000, 0

(Rappelons que HEX$(300) = 12C, HEX$(120) = 78, HEX$(5000) = 1388 et que HEX$(0) = 0.)

Ces paramètres se présenteront ainsi en mémoire :



Si nous voulons par exemple charger le registre DE avec le troisième paramètre, en l'occurrence 5000 (&1388), il nous faut charger l'octet fort dans D et l'octet faible dans E :

(IX + 3) dans D et (IX + 2) dans E

Précisons enfin que le nombre de paramètres transmis est chargé, lui, dans le registre A (accumulateur) au moment de l'appel du programme. Nous verrons que cela peut être très utile.

LES PRÉCAUTIONS INDISPENSABLES

Le langage machine n'est pas d'un maniement aussi aisé que le BASIC, et si vous ne voulez pas être rapidement et à tout jamais découragé, vous ferez bien de tenir compte des conseils suivants :

- N'oubliez pas de protéger votre zone d'implantation grâce à l'instruction MEMORY. De plus, faites bien attention que votre programme ne "déborde" pas de l'adresse 43903. Évitez également d'implanter vos programmes dans des zones inférieures à 16384, cela vous épargnera des désagréments dont il serait trop long de parler ici.
- Enregistrez *systématiquement* vos programmes avant de les essayer. Le langage machine réagit parfois extrêmement mal aux erreurs. Une seule petite faute, et vous risquez de voir le programme que vous avez mis deux heures à rentrer s'atomiser et se perdre définitivement

dans les entrailles de la machine (en tout état de cause, et quoi que vous fassiez, sachez quand même que vous ne risquez en aucun cas de détraquer votre ordinateur).

Il arrive souvent, lorsqu'un programme se "plante", que rien ne se passe et que le clavier ne réagisse plus. Dites-vous bien qu'il s'agit *toujours* d'une erreur de conception (inutile d'accuser la machine, elle ne fait jamais d'erreur), et que l'ordinateur doit être branché sur une boucle malsaine dont il ne peut plus sortir. Dans ce cas-là, une seule solution : éteindre et rallumer.

- Tenez la pile sous haute surveillance. Comme il a été dit précédemment, la pile est souvent un moyen commode pour stocker des données. Mais n'oubliez pas que l'ordinateur s'en sert également de son côté, et voici de quelle manière :

Nous avons déjà comparé l'instruction CALL à l'instruction GOSUB, la seule différence étant que la première appelle un sous-programme en langage machine, et la seconde un sous-programme en BASIC.

De la même manière qu'un sous-programme en BASIC doit toujours se terminer par un RETURN, un programme en langage machine doit toujours se terminer par une instruction de retour. Dès qu'une telle instruction est rencontrée, l'ordinateur quitte le langage machine et retourne exécuter l'instruction située immédiatement derrière l'instruction CALL, puis poursuit normalement le programme BASIC. Cela n'est possible que parce que, au moment où il a rencontré l'instruction CALL, il a stocké l'adresse de la suivante sur la pile, en prévision du retour.

Cette adresse, il s'attend donc à la retrouver sur la pile au moment du retour. Si un "tripotage" intempestif de la pile l'a entre-temps dénaturée sans la remettre en état, l'ordinateur récupérera une donnée fausse et le programme va sauter n'importe où. La règle d'or est donc: *Au moment du retour d'un sous-programme en langage machine, la pile doit être dans le même état qu'au moment de l'appel de ce sous-programme.*

Vous verrez que cette règle n'est pas très difficile à appliquer pour peu que l'on fasse preuve d'un minimum de rigueur et de minutie.

Lorsque j'utilise beaucoup la pile, je me sers pour ma part d'un petit "truc" aussi simple qu'efficace : je manipule physiquement une pile faite de petits cartons que j'entasse les uns sur les autres, ou au contraire que j'enlève de la pile en même temps que mon programme. Le premier carton à être systématiquement déposé est bien sûr intitulé "adresse de retour du programme". Quand le programme s'achève, c'est-à-dire au moment où l'instruction de retour au BASIC est rencontrée, il ne doit plus rester d'autre carton au-dessus de celui-là. Si c'est le cas,

je rajoute, avant l'instruction de retour, les instructions nécessaires pour les faire disparaître.

Cette histoire de pile est réellement très importante. Combien de débutants se sont arraché les cheveux devant des programmes qui se plantaient, alors qu'il s'agissait tout simplement d'un problème de pile !

C'est sur ce dernier conseil que s'achèvent les deux premières parties. Vous en savez maintenant assez pour vous attaquer au premier programme.

III
EXEMPLES DE PROGRAMMES

1. PAUSE DE X SECONDES

Nous allons, pour votre baptême du feu, nous attaquer à un programme bien modeste, puisqu'il ne comprend que 23 octets. Il permettra d'effectuer des pauses de longueur variable, et remplacera avantageusement les boucles de temporisation laborieuses et imprécises que l'on est obligé d'utiliser sur l'Amstrad.

Ce programme débute à l'adresse 43880, finit en 43902 (chiffre de vérification : 2976), est relogeable (ce qui veut dire que l'on peut l'implanter ailleurs en mémoire si l'on veut ce qui n'est pas le cas de tous les programmes, nous le verrons plus tard), et son format d'appel est le suivant :

CALL 43880,X

Le paramètre X détermine, en secondes, la longueur de pause désirée. Voici les conventions de la présentation du listing :
La première colonne indique des numéros de ligne qui ne servent à rien d'autre qu'aux explications. La deuxième indique l'adresse du premier octet du code machine de cette ligne (cette adresse est également fournie en hexadécimal si, pour une raison ou pour une autre, elle est remarquable). La troisième colonne, sans titre, sert à visualiser les sauts éventuels. La quatrième indique les instructions du programme (en hexadécimal, bien sûr), et la cinquième les mnémoniques de ces mêmes instructions (les mnémoniques sont simplement des représentations symboliques des différentes instructions, dont il est plus facile de se souvenir que des codes machine).

Rappelez-vous enfin que toutes ces instructions sont décrites en détail, le cas échéant avec leurs variantes, dans la Cinquième Partie. N'hésitez pas à vous y reporter si besoin est.

LIGNE	ADRESSE DU 1er CODE DE CETTE LIGNE DEC.	HEXA.	CODE-MACHINE	MNEMONIQUE
1	43880		DD,66,1	LD H, (IX+1)
2	43883		DD,6E,0	LD L, (IX+0)
3	43886		29	ADD HL,HL
4	43887		E5	PUSH HL
5	43888		21,56,FE	LD HL,65110
6	43891		2B	DEC HL
7	43892		7C	LD A,H
8	43893		B5	OR L
9	43894		20,FB	JR NZ,-5
10	43896		E1	POP HL
11	43897		2B	DEC HL
12	43898		7C	LD A,H
13	43899		B5	OR L
14	43900		20,F1	JR NZ,-15
15	43902		C9	RET

Le principe de ce programme est exactement le même que celui d'une boucle de temporisation en BASIC : nous allons demander à l'ordinateur... de ne rien faire, mais un grand nombre de fois, pour faire "passer le temps".

En réalité, nous nous servirons de deux boucles, pour une raison que vous comprendrez plus loin. Nous allons tout d'abord nous intéresser à la boucle constituée par les lignes 5 à 9, qui dure exactement une demi-seconde :

Ligne 5

Chargement du registre HL avec le nombre 65110 (&FE56). Vous remarquerez qu'après le code de chargement (&21), on met *d'abord l'octet faible* de la valeur à charger, et ensuite l'octet fort (signalons également que pour charger un registre double, l'octet fort doit toujours être indiqué, même s'il est égal à 0). Pourquoi ce nombre 65110 ? Il a été déterminé par essais successifs, de manière que cette boucle centrale dure une demi-seconde (à quelques millièmes près).

Ligne 6

Décrémentation du registre HL, ce qui signifie que l'on enlève 1 au contenu de ce registre.

Ligne 7

Chargement du registre A (ou accumulateur) avec le contenu du registre H (donc avec l'octet fort du contenu de HL).

Remarque

Le contenu de H n'est pas modifié après l'instruction. D'une manière générale, les instructions de chargement n'affectent jamais le contenu de la source.

Ligne 8

Littéralement "Exécution d'un OU logique entre le registre L et le registre A". Il faut savoir que le résultat d'un OU logique entre un registre et l'accumulateur n'est égal à 0 *que si* le contenu de ces deux registres est égal à 0.

Comme A est chargé avec le contenu de H, c'est exactement comme si nous faisions un OU logique entre H et L (ce qui n'est pas possible directement). Cette ligne 8 aura donc pour résultat 0 quand *et seulement quand* H et L seront égaux à 0, donc quand HL sera vide.

Ligne 9

Elle signifie "Saut relatif de −5 si non nul", c'est-à-dire tant que l'indicateur Z (voir Première Partie, section 5) n'est pas mis, signalant que l'opération qui vient d'avoir lieu (le OU logique) avait pour résultat 0.

Pour résumer, le saut aura lieu tant que HL ne sera pas vide (saut de −5, donc en ligne 6 ; voir à ce sujet l'Annexe I qui explique tout sur les sauts relatifs). Si HL est vide, le saut sera ignoré et le programme se poursuivra en ligne 10.

La boucle est donc bouclée : en ligne 6, HL est à nouveau décrémenté, puis A est chargé avec H, le OU logique effectué, et ainsi de suite, jusqu'à ce que HL ait été décrémenté 65110 fois. L'ensemble est donc exécuté en une demi-seconde, ce qui prouve bien la rapidité du langage.

Il reste maintenant à étudier la boucle générale.

Nous avons dit que, lors de l'appel du sous-programme, le paramètre X était sensé exprimer un temps en secondes. Or, notre boucle centrale ne fait qu'une demi-seconde. La première chose qu'il y a donc à faire, est de multiplier le paramètre X par 2 :

Ligne 1

Si vous avez bien lu la section "Les paramètres transmissibles", cette ligne ne devrait pas poser de problème : chargement du registre H avec le contenu de l'emplacement mémoire d'adresse IX+1, c'est-à-dire avec l'octet fort de X.

Ligne 2

Chargement du registre L avec le contenu de l'emplacement mémoire d'adresse IX+0, c'est-à-dire avec l'octet faible de X. Après ces deux lignes, HL est donc chargé avec X.

Ligne 3

Addition de HL... avec lui-même ! Le résultat étant chargé dans HL, cela revient donc à multiplier par 2 son contenu. Il est maintenant chargé avec 2 * X.

Ligne 4

Empilement du registre HL : le contenu du registre HL est déposé sur la fameuse pile évoquée dans la Première Partie. Le registre HL va en effet être utilisé pour autre chose, mais nous aurons encore besoin de son contenu, que nous stockons donc en le déposant sur la pile.

Remarque importante

Lorsque l'on empile un registre, celui-ci ne se vide pas, contrairement à ce que l'on pourrait croire. Après cette ligne 4, par exemple, HL contient toujours X * 2. L'empilement provoque en quelque sorte un recopiage de la valeur sur la pile.

Après cette ligne, le programme arrive à la ligne 4, où commence la boucle centrale déjà étudiée. Cette boucle est exécutée, après quoi le programme passe à la ligne 10.

Ligne 10

Dépilement du registre HL : la donnée sauvegardée en ligne 4 est retirée de la pile et chargée dans le registre HL, qui contient donc maintenant à nouveau X * 2.

Lignes 11, 12, 13

Elles suivent exactement le même processus que les lignes 6, 7 et 8 déjà étudiées : décrémentation de HL, chargement de A avec H, OU logique entre A et L. Comme vous l'avez sans doute déjà compris, il s'agit là de notre boucle générale, ou tout au moins de sa partie compteur.

Ligne 14

Saut relatif de –15 (en ligne 4) si non nul : ni HL n'est pas encore égal à 0, le programme saute en ligne 4 où le contenu actuel de HL est

de nouveau "mis au frais". Ensuite, nouveau départ de la boucle centrale pour une demi-seconde.

L'ensemble du processus se renouvellera (X * 2) fois, et comme la boucle centrale dure une demi-seconde, le temps total sera bien de X secondes. Lorsque le compteur de la boucle générale atteint 0, le saut n'est pas effectué et le programme passe à la ligne 15.

Ligne 15

Cette instruction signale la fin du programme en langage machine et provoque le retour au BASIC.

Le programme doit être implanté selon les principes exposés dans la Deuxième Partie et peut être ensuite appelé à partir du BASIC :

```
100 CALL 43880,10
```

Cette ligne BASIC provoquera une pause de dix secondes.

Signalons qu'il est facile de modifier le programme pour que le paramètre X détermine un temps exprimé en fractions de seconde. Il suffit pour cela de jouer sur la valeur que l'on charge dans HL à la ligne 5 (avec une valeur de 6290, par exemple, un CALL 43880,1 provoquera une pause de 1/10 de seconde).

En procédant par tâtonnements, on peut ainsi trouver n'importe quelle unité de pause. Il faut bien sûr pour cela utiliser l'instruction BASIC TIME :

```
100 TE=TIME/300 : CALL 43880,1 : TE1=TIME/300 :
PRINT TE1-TE
```

Le temps de pause s'affichera en secondes.

2. INTÉGRATION D'UN PROGRAMME MACHINE

Pour simplifier la manipulation des programmes en langage machine, et pour rendre leur emploi moins dangereux (on a vite fait de taper 43800 au lieu de 43880, par exemple), il est possible d'effectuer ce que l'on appelle une *intégration*. Cela permet de donner à ces programmes des noms, grâce auxquels il sera possible de les appeler en BASIC, comme s'il s'agissait véritablement de nouvelles instructions.

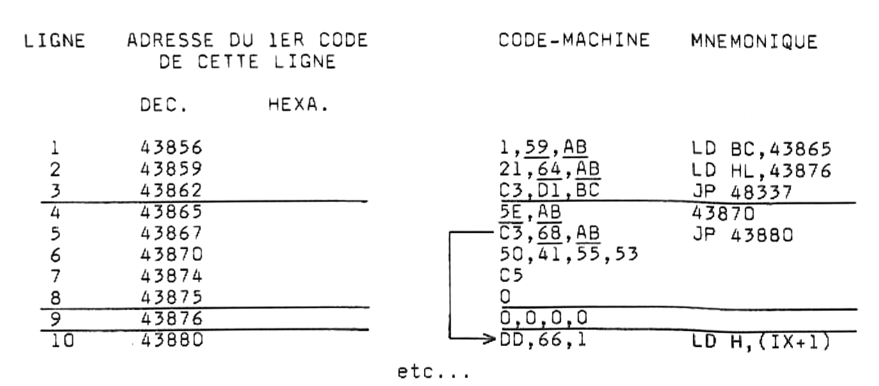
Pour illustrer la méthode d'intégration, nous allons nous servir du programme précédent et nous arranger pour qu'il soit possible de l'appeler avec le format suivant :

I PAUSE,X

Notons que l'extension d'instruction est signalée par un trait vertical (SHIFT + @) placé devant le mot.

Aucune autre intégration ne sera effectuée pour les programmes suivants, mais vous pourrez très facilement les faire vous-même en vous servant de ce modèle.

Voici le listing du programme d'intégration :



LIGNE	ADRESSE DU 1ER CODE DE CETTE LIGNE		CODE-MACHINE	MNEMONIQUE
	DEC.	HEXA.		
1	43856		1,59,AB	LD BC,43865
2	43859		21,64,AB	LD HL,43876
3	43862		C3,D1,BC	JP 48337
4	43865		5E,AB	43870
5	43867		C3,68,AB	JP 43880
6	43870		50,41,55,53	
7	43874		C5	
8	43875		0	
9	43876		0,0,0,0	
10	43880		DD,66,1	LD H,(IX+1)

etc...

Le programme peut être divisié en quatre parties :

- Lignes 1, 2 et 3 : Intégration proprement dite.
- Lignes 4 à 8 : Table d'instruction (nous verrons de quoi il s'agit plus loin).
- Ligne 9 : Espace réservé pour la machine.
- Lignes 10 et suivantes : Programme pause identique au précédent.

Nous allons étudier tout cela dans le désordre pour que, paradoxalement, cela soit plus clair.

Ligne 6

Il s'agit de la représentation hexadécimale des codes ASCII (à ce sujet, voir page A3.1 du guide de l'utilisateur) des lettres du mot choisi pour l'instruction, *sauf la dernière* (en l'occurrence, il s'agit donc des codes de P, de A, de U, et de S).

Ligne 7

Représentation hexadécimale du code ASCII de la dernière lettre du mot d'instruction *plus &80* (en l'occurrence, il s'agit donc du code ASCII de E, plus &80, ou plus 128 en décimal).

Ligne 8

Le 0 signifie à la machine que le mot est terminé.

Ligne 9

Ces quatre octets sont réservés pour la machine (et plus exactement pour un système appelé Kernel) ils lui serviront à gérer l'extension. La manière dont elle s'y prendra nous importe peu.

Ligne 4

Il ne s'agit pas là d'une instruction, mais tout simplement de l'adresse en mémoire du code ASCII de la première lettre du mot d'instruction (ici &50, code ASCII de P, à l'adresse 43870).

Ligne 5

L'instruction C3 est une instruction de saut que l'on peut comparer au GOTO BASIC. Cette instruction doit être suivie d'une adresse en mémoire (ici 43880 ou &AB68).

Remarque importante

D'une manière générale, lorsque l'on écrit une adresse ou une donnée quelconque dans un programme en langage machine, *l'octet faible est toujours écrit avant l'octet fort.*

Pour en revenir à notre ligne 5, il s'agit donc d'un saut à l'adresse 43880, à laquelle débute le programme de pause proprement dit, tel

que nous l'avons déjà étudié. Cette ligne constitue le début de la table d'instruction.

Voyons maintenant l'intégration :

Ligne 1

Chargement, dans BC, de l'adresse du début de la table d'instruction (donc de l'adresse du premier code de la ligne 5).

Ligne 2

Chargement, dans HL, de l'adresse du premier des quatre octets réservés.

Ligne 3

Saut à l'adresse 48337. A cette adresse, qui fait partie du système interne de la machine, se trouve un programme que l'on pourrait appeler "Intégrer une instruction". Là encore, nous laisserons l'ordinateur faire son travail, sans nous préoccuper de la manière dont cela se passe.

En réalité, on ne peut comprendre parfaitement ce programme à moins de connaître à fond le programme interne d'intégration.

Quant à nous, il nous suffit de savoir que ça marche...

Une fois le programme chargé en mémoire, *il doit être initialisé une fois et une seule* en faisant CALL 43856.

On peut ensuite utiliser le programme de pause grâce à l PAUSE,X. En cas d'écriture erronée de l'instruction, l'éditeur sortira en principe un *syntax error*.

Remarque

Contrairement au programme précédent, celui-ci n'est pas directement relogeable. Il comprend en effet ce que l'on appelle des adresses internes ; la première ligne, par exemple, charge BC avec l'adresse &AB59 (43865), qui est une adresse du programme lui-même. Si vous souhaitez déplacer ce programme et le mettre ailleurs en mémoire, ce qui est toujours possible, il ne faut surtout pas oublier que cette adresse va également changer en fonction de la nouvelle place du programme, et qu'il faut donc modifier la ligne 1, ainsi que toutes les lignes comprenant des adresses internes.

Pour faciliter les déplacements de programme que vous pourriez entreprendre, les adresses à modifier seront toujours soulignées dans les listings.

3. DESSIN D'UN QUADRILATÈRE

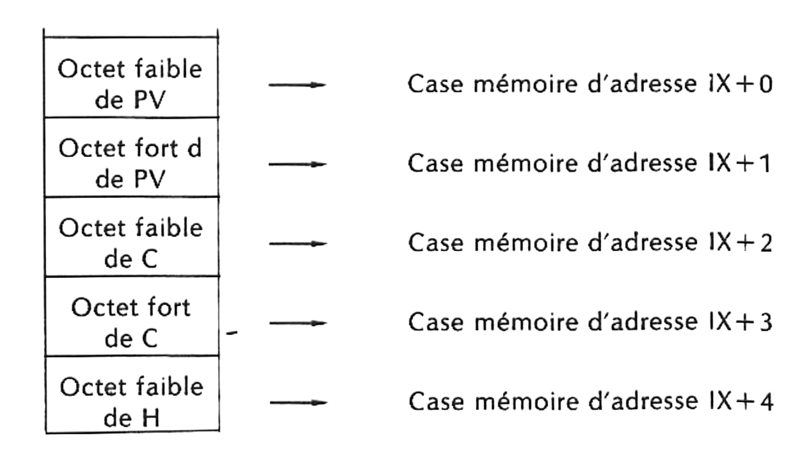
Le programme que nous allons étudier maintenant est un peu plus ambitieux : il permettra en effet la réalisation de quadrilatères de tailles, couleurs et localisation différentes.

Il débute à l'adresse 43790, finit en 43902 (chiffre de vérification : 14746), n'est pas directement relogeable et son appel aura le format suivant :

CALL 43790,XC,YC,L,H,C,PV

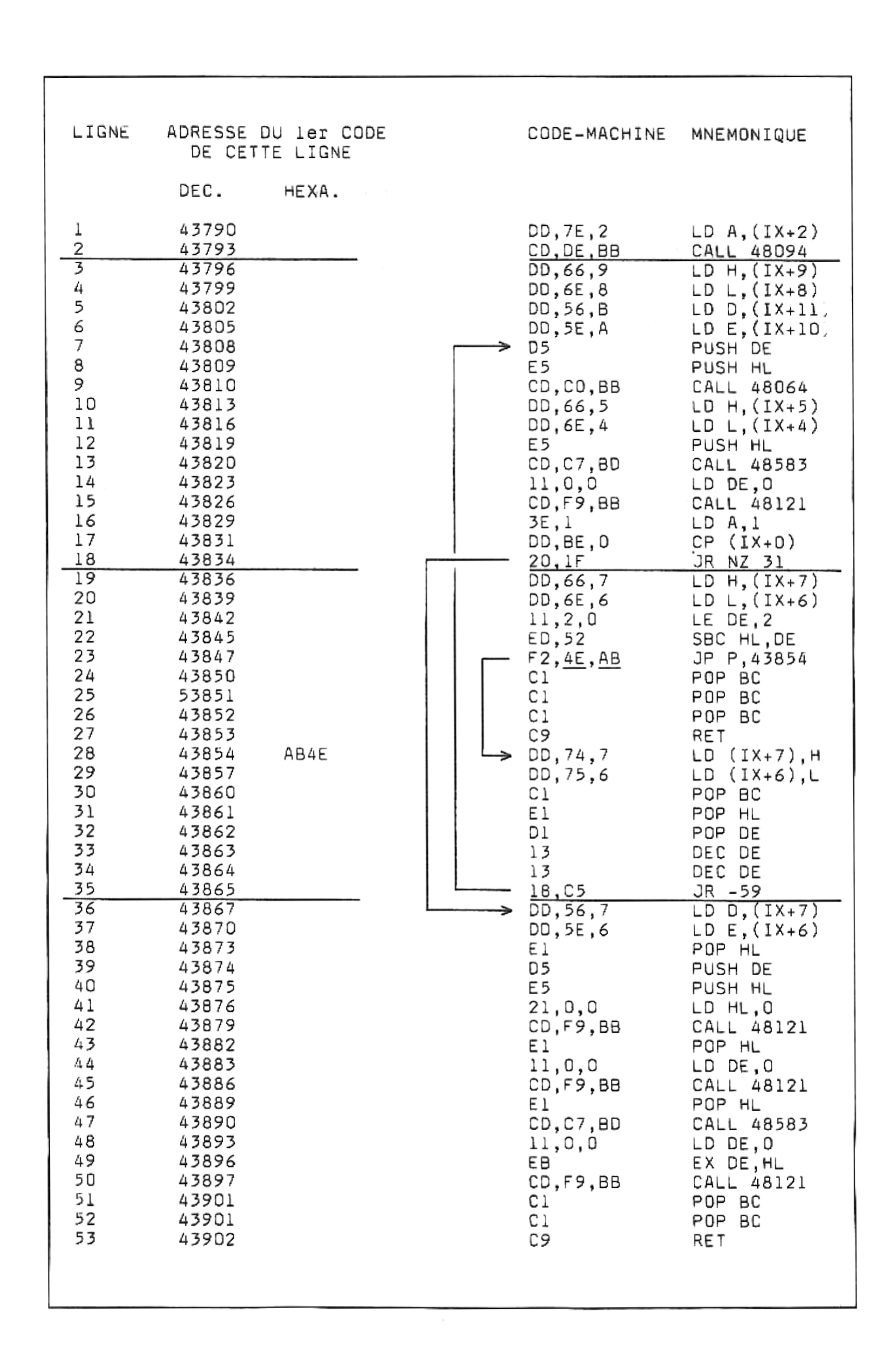
XC et YC sont les coordonnées du coin supérieur gauche du quadrilatère, L détermine sa largeur, H sa hauteur, C sa couleur et PV indique s'il doit être plein ou vide (1 : plein, 0 : vide).

Pour qu'il n'y ait plus de doute quant à la situation de ces paramètres au moment de l'appel du programme, rappelons une dernière fois le pointage du registre IX :



Octet fort de H	⟶	Case mémoire d'adresse IX+5
Octet faible de L	⟶	Case mémoire déadresse IX+6
Octet fort de L	⟶	Case mémoire d'adresse IX+7
Octet faible de YC	⟶	Case mémoire d'adresse IX+8
Octet fort de YC	⟶	Case mémoire d'adresse IX+9
Octet faible de XC	⟶	Case mémoire d'adresse IX+10
Octet fort de XC	⟶	Case mémoire d'adresse IX+11

Attention, la lecture de l'Annexe II sur les coordonnées est un préalable indispensable à l'étude de ce programme, dont voici le listing :



LIGNE	ADRESSE DU 1er CODE DE CETTE LIGNE		CODE-MACHINE	MNEMONIQUE
	DEC.	HEXA.		
1	43790		DD,7E,2	LD A,(IX+2)
2	43793		CD,DE,BB	CALL 48094
3	43796		DD,66,9	LD H,(IX+9)
4	43799		DD,6E,8	LD L,(IX+8)
5	43802		DD,56,B	LD D,(IX+11)
6	43805		DD,5E,A	LD E,(IX+10)
7	43808		D5	PUSH DE
8	43809		E5	PUSH HL
9	43810		CD,C0,BB	CALL 48064
10	43813		DD,66,5	LD H,(IX+5)
11	43816		DD,6E,4	LD L,(IX+4)
12	43819		E5	PUSH HL
13	43820		CD,C7,BD	CALL 48583
14	43823		11,0,0	LD DE,0
15	43826		CD,F9,BB	CALL 48121
16	43829		3E,1	LD A,1
17	43831		DD,BE,0	CP (IX+0)
18	43834		20,1F	JR NZ 31
19	43836		DD,66,7	LD H,(IX+7)
20	43839		DD,6E,6	LD L,(IX+6)
21	43842		11,2,0	LE DE,2
22	43845		ED,52	SBC HL,DE
23	43847		F2,4E,AB	JP P,43854
24	43850		C1	POP BC
25	53851		C1	POP BC
26	43852		C1	POP BC
27	43853		C9	RET
28	43854	AB4E	DD,74,7	LD (IX+7),H
29	43857		DD,75,6	LD (IX+6),L
30	43860		C1	POP BC
31	43861		E1	POP HL
32	43862		D1	POP DE
33	43863		13	DEC DE
34	43864		13	DEC DE
35	43865		18,C5	JR -59
36	43867		DD,56,7	LD D,(IX+7)
37	43870		DD,5E,6	LD E,(IX+6)
38	43873		E1	POP HL
39	43874		D5	PUSH DE
40	43875		E5	PUSH HL
41	43876		21,0,0	LD HL,0
42	43879		CD,F9,BB	CALL 48121
43	43882		E1	POP HL
44	43883		11,0,0	LD DE,0
45	43886		CD,F9,BB	CALL 48121
46	43889		E1	POP HL
47	43890		CD,C7,BD	CALL 48583
48	43893		11,0,0	LD DE,0
49	43896		EB	EX DE,HL
50	43897		CD,F9,BB	CALL 48121
51	43901		C1	POP BC
52	43901		C1	POP BC
53	43902		C9	RET

Comme vous le voyez sur le listing, le programme peut être divisé en quatre blocs :

- Le premier n'est composé que de deux lignes et fixe la couleur.
- Le deuxième va des lignes 3 à 18. Il dessine la première ligne du quadrilatère puis effectue le branchement à telle ou telle partie du programme selon que le quadrilatère doit être plein ou vide.
- Le troisième va des lignes 19 à 35 et dessine un quadrilatère plein.
- Le quatrième, enfin, dessine un quadrilatère vide.

Ligne 1

Chargement de A avec l'octet faible du paramètre C (le numéro de couleur étant forcément compris entre 0 et 15, l'octet fort de sa représentation hexadécimale sera toujours égal à 0, et nous ne nous en préoccuperons pas. C'est d'ailleurs pour cela que nous pouvons utiliser un registre simple).

Ligne 2

Appel de la routine du système d'exploitation d'adresse &BBDE. Cette routine fixe la couleur d'écriture graphique (avant l'appel de cette routine, A doit être chargé avec le numéro de couleur souhaitée, ce qui a bien été fait à la ligne précédente).

Notons que l'instruction &CD est tout à fait comparable à l'instruction BASIC GOSUB : il s'agit bien de l'appel d'un sous-programme. Une fois ce sous-programme terminé, le retour sera effectué, en l'occurrence à la ligne 3.

Lignes 3 et 4

Chargement de HL avec l'ordonnée (YC) du coin supérieur gauche du quadrilatère désiré.

Lignes 5 et 6

Chargement de DE avec l'abscisse (XC) du même point.

Ligne 7

Empilement de DE. Nous aurons en effet à nous resservir de cette valeur XC, et il est plus facile de charger un registre à partir de la pile (1 octet suffit), qu'à partir de IX (6 octets sont nécessaires).

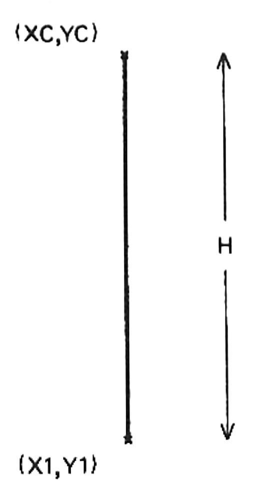
Ligne 8

Empilement de HL pour la même raison. (La pile va être souvent sollicitée, et il est vivement conseillé de noter au fur et à mesure les dépilements et empilements).

Ligne 9

Appel de la routine d'adresse &BBC0, qui fixe le curseur graphique à une position absolue dont les coordonnées doivent être, avant l'appel, dans DE pour l'abscisse et dans HL pour l'ordonnée. Le curseur est donc maintenant fixé sur le coin supérieur gauche de notre futur quadrilatère.

Avant de passer aux lignes suivantes, un dessin nous permettra d'avoir une vue plus claire du processus :



Nous avons, en ligne 9, fixé le curseur sur (XC,YC), et nous voulons maintenant tracer la ligne allant de (XC,YC) à (X1,Y1). *Relativement au point (XC,YC),* X1 = 0 et Y1 = −H.

Nous allons donc commencer par charger ces valeurs dans DE et HL.

Lignes 10 et 11

Chargement de HL avec le paramètre H.

Ligne 12

Sauvegarde de cette valeur par empilement.

Ligne 13

Appel de la routine d'adresse &BDC7, qui a pour fonction d'inverser le signe du contenu de HL. Ce dernier est donc maintenant chargé avec −H.

Ligne 14

Chargement du registre DE avec 0 (n'oublions pas que, pour charger un registre double, il faut obligatoirement indiquer les deux octets de la valeur, le fort et le faible, même s'ils sont égaux à 0).

Ligne 15

Appel de la routine d'adresse &BBF9, qui trace une ligne à partir de la position actuelle du curseur jusqu'à une position relative déterminée par DE (abscisse) et HL (ordonnée).

Voilà donc notre première ligne tracée (de plus, le curseur se trouve maintenant en X1,Y1).

Il est temps maintenant de tester le paramètre PV pour déterminer si le quadrilatère doit être plein ou vide.

Ligne 16

Chargement de A avec 1 (nous verrons pourquoi plus loin).

Ligne 17

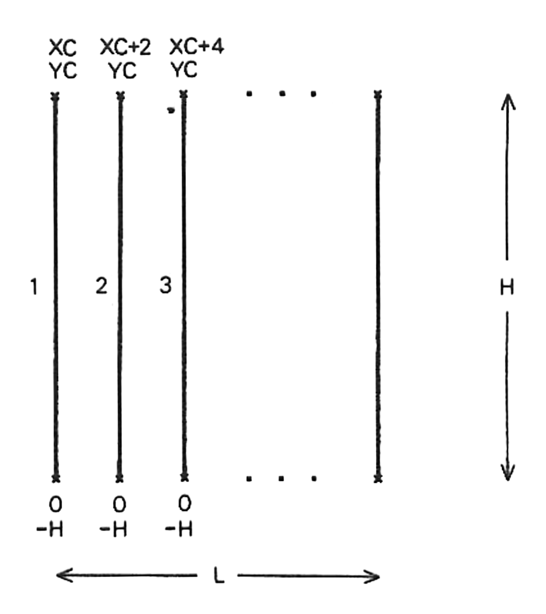
Comparaison de A et du contenu de l'emplacement d'adresse IX+0. La comparaison se passe de la manière suivante : le contenu de l'emplacement mémoire d'adresse IX+0 est soustrait de A, et le résultat n'est pas conservé (ce qui revient à dire que A n'est pas modifié. C'est en quelque sorte une soustraction "de tête"). Néanmoins, les indicateurs Z et S sont modifiés en fonction : en particulier, Z est mis à 1 si la comparaison indique l'égalité (en l'occurrence, si (IX+0)=A, ce dernier ayant été chargé avec 1).

Pour résumer, l'indicateur de 0 sera mis si le paramètre PV=1.

Ligne 18

Saut relatif de 31 (en ligne 36), si l'indicateur de 0 n'est pas mis, donc s'il s'agit de dessiner un carré vide. Dans le cas contraire, le saut n'est pas fait et le programme continue en ligne 19, où commence l'exécution du quadrilatère plein.

La figure ci-après montre le principe adopté :



Il s'agit de dessiner une série de lignes les unes contre les autres, de manière à obtenir un quadrilatère plein. Chacune d'entre elles sera tracée exactement de la même manière que la première, à la seule différence que les abscisses de l'extrémité supérieure seront, pour chaque nouvelle ligne, décalées vers la droite. Comme vous le voyez, nous avons indiqué ces abscisses sous la forme : XC + 2, XC + 4, etc., ce qui signifie que ce décalage se fera de deux en deux. En voici la raison : de par le fonctionnement de l'écran graphique du CPC, une ligne verticale a obligatoirement une largeur de 1 pixel en mode 2, de 2 pixels en mode 1, et de 4 pixels en mode 0. Puisque nous avons choisi de travailler en mode 1, chaque décalage vers la droite pourra être de 2 pixels (nous pourrions faire des décalages de 1, mais le temps d'exécution serait alors doublé inutilement car nous écririons sur des lignes déjà tracées).

Pour en revenir à la figure, les coordonnées du point bas de chaque ligne sont bien sûr relatives au point haut de la même ligne. Puisque nous allons tracer toutes ces lignes en suivant le même processus, il va nous falloir une boucle. Or, qui dit boucle dit compteur.

C'est le paramètre L (largeur) qui va nous en tenir lieu : nous voulons un quadrilatère de largeur L (L étant exprimé en pixels) ; or, chaque fois que nous traçons une ligne, elle fait 2 pixels de large. Chaque fois, donc, nous enlèverons 2 à la largeur, jusqu'à ce que le résultat soit négatif (on ne peut pas dire : "jusqu'à ce que le résultat soit nul" car, si L est impair, le compteur passerait de 1 à −1 et ne serait donc

jamais nul : cas typique d'une boucle sans fin qui peut obliger à éteindre la machine).

Le compteur est constitué par les lignes 19 à 23 :

Lignes 19 et 20 :

HL est chargé avec le paramètre L.

Ligne 21

Chargement de DE avec 2.

Ligne 22

DE est soustrait de HL (le résultat est rangé dans HL).

Ligne 23

Saut conditionnel à l'adresse 43854 (&AB4E, ligne 28) si le résultat est *positif*, c'est-à-dire si la boucle doit continuer.

Nous allons laisser de côté les lignes 24 à 37 pour le moment, et nous intéresser d'abord au cas où la boucle n'est pas terminée :

Lignes 28 et 29

Le contenu de H est chargé dans l'emplacement mémoire d'adresse IX+7 et celui de L dans l'emplacement mémoire d'adresse IX+6. Le résultat est donc qu'à chaque tour de boucle la nouvelle valeur de L (qui, ne l'oublions pas, vient d'être décrémentée de 2 à la ligne 22) est rangée à la place de l'ancienne. A chaque tour également, elle sera reprise en lignes 18 et 19, puis diminuée de 2, puis rangée, et ainsi de suite jusqu'à ce qu'elle devienne négative.

Ligne 30

Si vous avez bien suivi la pile, vous savez que, jusqu'à présent, il y avait le paramètre H sur son sommet. Nous n'aurons plus besoin de cette donnée, et nous la faisons disparaître de la pile en dépilant BC.

Ligne 31

Sur le sommet de la pile se trouve maintenant l'ordonnée YC. Elle est chargée dans HL.

Ligne 32

Chargement de l'abscisse XC (à partir de maintenant, nous considérerons que vous connaissez, à chaque instant, l'état de la pile).

Lignes 33 et 34

Comme nous l'avons expliqué précédement, l'abscisse du point de départ de la ligne suivante doit être décalée de 2 vers la droite par rapport à celle qui vient d'être tracée. C'est pourquoi nous incrémentons (=ajoutons 1) deux fois DE.

HL est donc maintenant chargé avec YC, et DE avec XC+2 (tout au moins au deuxième tour de boucle. Au troisième tour ce sera XC+4, au quatrième tour XC+6, etc.).

Ligne 35

Saut relatif de −59, c'est-à-dire en ligne 7. Là, les valeurs contenues actuellement dans DE et HL sont sauvegardées, puis le curseur est fixé, la ligne tracée, et ainsi de suite.

Lorsque la boucle est terminée, le saut de la ligne 23 ne se fait pas, et le programme passe à la ligne 24.

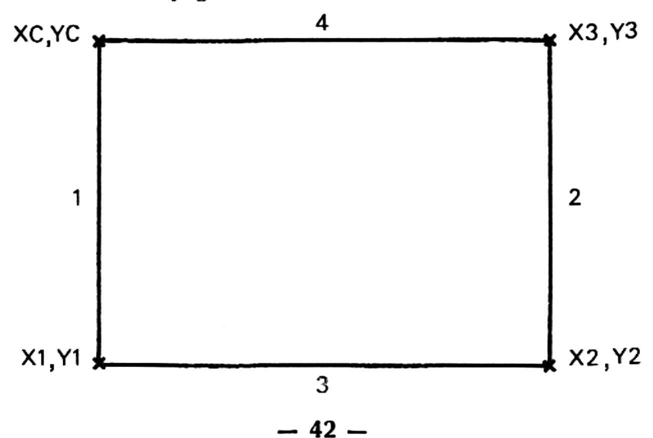
Lignes 24, 25 et 26

Si le programme arrive à la ligne 24, c'est donc que le dessin du quadrilatère est terminé, et qu'il faut revenir au BASIC. Mais auparavant, *il faut remettre la pile en état* (revoir à ce sujet la Deuxième partie).

Nous avions empilé trois valeurs sur la pile, elles y sont encore pour l'instant. La pile est donc "nettoyée" par trois dépilements successifs de BC (nous aurions pu dépiler n'importe quel registre, mais dans ces cas-là, et pour des raisons qu'il serait trop long de développer, il faut mieux utiliser BC).

L'étude du bloc "Quadrilatère plein" étant terminé, passons au bloc "Quadrilatère vide".

Le processus de traçage sera le suivant :



Les lignes 2, 3 et 4 seront tracées les unes après les autres et dans cet ordre. Il faut remarquer que :

- Les coordonnées du point (X2,Y2) relativement à (X1,Y1) sont (L,0).
- Les coordonnées du point (X3,Y3) relativement à (X2,Y2) sont (0,H).
- Les coordonnées du point (XC,YC) relativement à (X3,Y3) sont (−L,0).

(N'oublions pas que XC et YC sont des cordonnées absolues.)

Il est peut-être utile de rappeler la situation au moment où s'effectue le branchement de la ligne 18 :

La ligne 1 est tracée, le curseur est positionné en (X1,Y1) et il y a trois valeurs stockées sur la pile (H se trouve au sommet, YC juste en dessous et XC encore en dessous).

Lignes 36 et 37

Chargement de DE avec le paramètre L.

Lignes 38, 39 et 40

Il s'agit ici de sauvegarder le paramètre chargé dans DE, mais, pour des raisons pratiques qui s'expliqueront d'elles-mêmes plus loin, il faut que cette valeur ne soit pas sur le sommet de la pile, mais à l'*avant-dernier étage*. C'est pourquoi la ligne 38 "soulève" H, après quoi DE est empilé, puis H "reposé".

Ligne 41

Chargement de HL avec 0.

Ligne 42

Appel de la routine d'adresse &BBF9. La ligne 2 est maintenant tracée (par la même occasion, le curseur est en (X2,Y2)).

Lignes 43, 44 et 45

Chargement de HL avec H, de DE avec 0, et tracé de la ligne 3.

Lignes 46 et 47

Chargement de L dans HL, puis appel de la routine &BDC7 (déjà étudiée en ligne 13). HL est donc maintenant chargé avec −L.

Ligne 48

0 dans DE.

Ligne 49

Une nouvelle instruction : les contenus de HL et DE sont échangés.

Pour tracer la ligne 4, c'est en effet DE qui doit être chargé avec −L et HL avec 0.

Ligne 50

Tracé de la ligne 4.

Lignes 51 et 52

Remise en état de la pile.

Ligne 53

Retour au BASIC.

L'utilisation de ce programme ne pose en principe aucun problème, même si vous fournissez des paramètres erronés ou si vous en oubliez. Dans le pire des cas, rien ne se passera.

Voici un programme de démonstration :

```
100 MODE 0
110 FOR I=1 TO 100 : GOSUB 180
120 CALL 43790,A,B,C,D,E,0
130 NEXT I
140 FOR I=1 TO 100 : GOSUB 180
150 CALL 43790,A,B,C,D,E,1
160 NEXT I
170 END
180 A=INT(RND * 640) : B=INT(RND * 400) : C=INT(RND
    * 300) : D=INT(RND * 300) : E=INT(RND * 16) :
    RETURN
```

Essayez aussi en intercalant la ligne :

```
135 POKE 43863,&2B
```

Un bon exercice serait de chercher à comprendre la raison de l'étrange résultat provoqué par cette ligne...

4. DÉFILEMENT D'UNE LIGNE

Ce programme permettra d'effectuer des défilements (ou *scrollings*) de ligne vers la droite ou vers la gauche.
Il débute à l'adresse 43830, finit en 43902 (chiffre de vérification : 7424), est relogeable tel quel, et son format d'appel est le suivant :

CALL 43830, NL, S

NL est le numéro de la ligne concernée et S détermine le sens de défilement (0 provoquera un défilement vers la gauche et 1 un défilement vers la droite).

Attention, il est absolument nécessaire de lire l'Annexe III, concernant la mémoire écran, avant d'aborder l'étude de ce programme.

Pour ce programme, nous allons commencer par une description globale de la démarche adoptée. Nous nous aiderons pour cela de la figure ci-après, qui représente une partie de la carte mémoire de la ligne n° 1 (cette ligne n'est bien sûr prise que comme exemple et le raisonnement reste le même quelle que soit la ligne choisie).

HL HL HL HL HL HL
DE DE DE DE DE DE

49152	49153	49154	.....	49229	49230	49231	1er trait
51200	51201		.....		51278	51279	2ème trait
53248	53249		.....		53326	53327	3ème trait
55296			.....			55375	4ème trait
57344			.....			57423	5ème trait
59392			.....			59471	6ème trait
61440			.....			61519	7ème trait
63448			.....			63567	8ème trait

Ne vous préoccupez pas, pour l'instant, des registres représentés.

Le moyen le plus simple de faire défiler cette ligne, par exemple vers la gauche, serait le suivant :

Transférer le contenu de l'emplacement mémoire 49153 dans l'emplacement mémoire 49152 (on peut dire transférer (49153) dans 49152), puis transférer (49154) dans 49153, puis (49155) dans 49154, et ainsi de suite jusqu'à la fin du premier trait qui se retrouve ainsi décalé d'un octet vers la gauche. Il suffit ensuite de renouveler l'opération avec les sept autres traits pour que la ligne entière se retrouve à son tour décalée d'une position vers la gauche.

Remarques

1. Il va nous falloir deux compteurs de boucle : un compteur que nous appellerons *compteur d'octets* et qui nous servira pour décaler les 80 octets d'un même trait, et un autre que nous appellerons *compteur général*, et qui comptera le nombre de traits qui ont été décalés, de manière que le programme s'arrête lorsque les huit traits le sont.

2. En ce qui concerne les octets des extrémités gauches de chaque trait (49152, 51200, 53248, etc.), nous avons dit que nous commencerions par transférer (49153) dans 49152. Mais il faut savoir qu'alors le contenu initial de 49152 sera perdu, puisque remplacé.

Or nous devons conserver ce contenu pour pouvoir le transférer quand le trait aura été entièrement décalé, en 49231.

Il faudra donc veiller à la sauvegarde du contenu des huit octets de gauche.

Le défilement vers la droite s'obtiendra évidemment selon le même principe, mais inversé : (49230) dans 49231, (49229) dans 49230, etc. Cette fois, ce sont les contenus des octets des extrémités droites de chaque trait qui devront être sauvegardés.

Pratiquement, les décalages seront obtenus grâce à deux instructions qui constitueront véritablement le cœur du programme.

Transfert répétitif de bloc avec décrémentation

Cette instruction sera utilisée pour les défilements vers la droite. Voici sa description, qu'il convient de lire calmement, plusieurs fois, et en s'aidant de la figure ci-après.

Le contenu de l'emplacement mémoire dont l'adresse est dans HL est chargé dans l'emplacement mémoire dont l'adresse est dans DE. Puis HL, DE et BC sont tous trois décrémentés (on leur enlève 1). Si BC est différent de 0, l'instruction est exécutée de nouveau. Sinon, le programme continue normalement.

La description est un peu rébarbative, mais en fait le fonctionnement de cette instruction est très simple. Regardez la figure suivante.

Imaginons qu'au départ HL soit chargé avec 49230, DE avec 49231, et BC avec 79. Lorsque l'instruction est exécutée, le contenu de 49230 est chargé dans 49231. Les trois registres sont décrémentés. HL est maintenant chargé avec 49229 (on dit aussi *pointé* sur 49229) et DE est pointé sur 49230. Quant à BC, qui contient maintenant 78, il est différent de 0. L'instruction est donc exécutée de nouveau, et ainsi de suite.

Ainsi, HL et DE vont se déplacer ensemble tout au long du trait, jusqu'au moment où BC sera égal à 0, auquel cas le transfert sera terminé (le compteur d'octets évoqué plus haut sera donc constitué par BC et initialisé avec 79).

En prenant le cas du premier trait, par exemple, signalons qu'à la fin de l'instruction HL sera "sorti" de l'écran, puisqu'il se retrouvera pointé sur 49151 alors que DE le sera sur 49152 (vous comprenez sans doute maintenant pourquoi BC est chargé avec 79 au lieu de 80 : ce dernier transfert ne nous intéresse pas).

Pour information, il faut également signaler que l'emplacement source, c'est-à-dire celui dont le contenu est transféré, n'est pas affecté par le transfert en question (il serait en fait plus juste de parler de recopiage plutôt que de transfert).

L'instruction suivante permettra le défilement vers la gauche.

Transfert répétitif de bloc avec incrémentation

Le contenu de l'emplacement mémoire dont l'adresse est dans HL est chargé dans l'emplacement mémoire dont l'adresse est dans DE. Les registres HL et DE sont ensuite *incrémentés* (on leur ajoute 1), tandis que BC est *décrémenté*. Tant que BC n'est pas égal à 0, l'instruction est exécutée une nouvelle fois.

Pour transférer le premier trait, par exemple, HL et DE devront donc être respectivement pointés sur 49153 et 49152.

Une dernière chose enfin : puisque nous appellerons le programme en ne fournissant que le numéro de ligne, il devra se charger de calculer les adresses avec lesquelles HL devra être initialisé (le deuxième octet du premier trait de cette ligne pour un défilement vers la gauche, et l'avant-dernier octet de ce même trait pour un défilement vers la droite). Le deuxième octet peut être calculé grâce à la formule suivante :

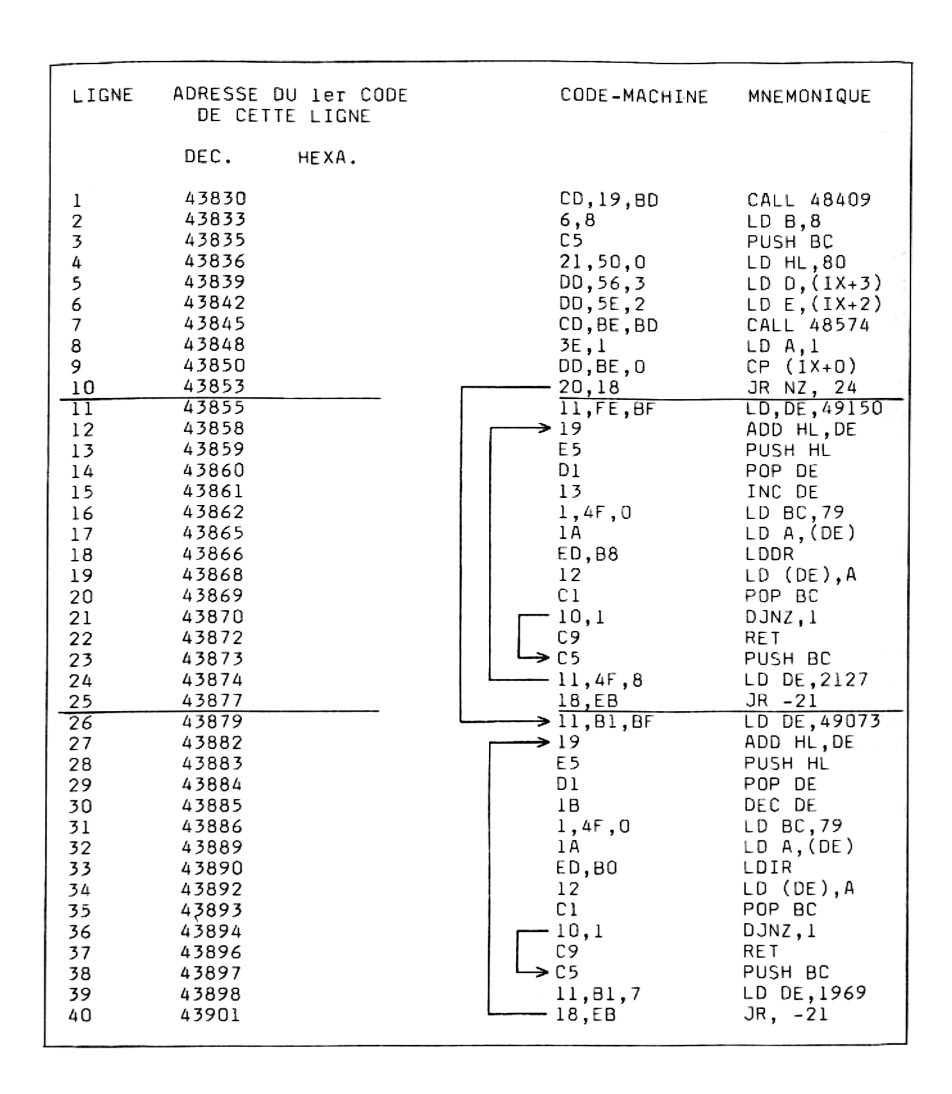
49150 + (80 * NL)

où NL est le numéro de ligne. L'avant dernier octet, lui, est donné par :

49073 + (80 * NL)

Vous pouvez vérifier, si vous en avez envie, que ces formules sont valables quel que soit le numéro de ligne.

Si tout cela est bien compris, nous pouvons maintenant en venir au programme, dont voici le listing :



LIGNE	ADRESSE DU 1er CODE DE CETTE LIGNE DEC.	HEXA.	CODE-MACHINE	MNEMONIQUE
1	43830		CD,19,BD	CALL 48409
2	43833		6,8	LD B,8
3	43835		C5	PUSH BC
4	43836		21,50,0	LD HL,80
5	43839		DD,56,3	LD D,(IX+3)
6	43842		DD,5E,2	LD E,(IX+2)
7	43845		CD,BE,BD	CALL 48574
8	43848		3E,1	LD A,1
9	43850		DD,BE,0	CP (IX+0)
10	43853		20,18	JR NZ, 24
11	43855		11,FE,BF	LD,DE,49150
12	43858		19	ADD HL,DE
13	43859		E5	PUSH HL
14	43860		D1	POP DE
15	43861		13	INC DE
16	43862		1,4F,0	LD BC,79
17	43865		1A	LD A,(DE)
18	43866		ED,B8	LDDR
19	43868		12	LD (DE),A
20	43869		C1	POP BC
21	43870		10,1	DJNZ,1
22	43872		C9	RET
23	43873		C5	PUSH BC
24	43874		11,4F,8	LD DE,2127
25	43877		18,EB	JR -21
26	43879		11,B1,BF	LD DE,49073
27	43882		19	ADD HL,DE
28	43883		E5	PUSH HL
29	43884		D1	POP DE
30	43885		1B	DEC DE
31	43886		1,4F,0	LD BC,79
32	43889		1A	LD A,(DE)
33	43890		ED,B0	LDIR
34	43892		12	LD (DE),A
35	43893		C1	POP BC
36	43894		10,1	DJNZ,1
37	43896		C9	RET
38	43897		C5	PUSH BC
39	43898		11,B1,7	LD DE,1969
40	43901		18,EB	JR, -21

Ligne 1

Appel de la routine d'adresse &BD19. Cette routine s'appelle "Attendre le retour du rayon" ; elle permet de synchroniser un programme d'animation d'écran avec le balayage de l'écran par le rayon, ce qui évite des effets optiques désagréables. Cette routine est souvent indispensable pour les animations, mais il faut savoir qu'elle ralentit considérablement l'exécution des programmes (lorsque vous aurez entré le programme, essayez par exemple de remplacer les trois codes de la première ligne par trois zéros).

Lignes 2 et 3

Chargement de 8 dans B, puis sauvegarde par empilement. Il s'agit bien sûr de la valeur initiale du compteur général (notons que malgré le "PUSH BC", la valeur contenue dans C nous importe peu).

Il faut maintenant aborder le calcul de l'octet de départ, en fonction de la ligne. Si vous regardez les deux formules proposées dans le préambule, vous pouvez constater que, quel que soit le sens de défilement choisi, elles comportent une partie commune : 80 * NL. Nous allons donc commencer par cela :

Ligne 4

HL est chargé avec 80.

Lignes 5 et 6

Chargement du paramètre NL (numéro de ligne) dans DE.

Ligne 7

Appel de la routine d'adresse &BDDE, qui multiplie HL par DE et range le résulat dans HL. Ce dernier est donc maintenant chargé avec 80 * NL.

Il est temps maintenant de déterminer le sens de défilement.

Ligne 8

A est chargé avec 1.

Lignes 9 et 10

Comparaison du paramètre S avec 1, puis saut relatif de 24 (en ligne 26) si non nul. Le branchement conditionnel est réalisé ici selon un processus déjà étudié dans le programme de dessin d'un quadrilatère (lignes 17 et 18). Seule la longueur du saut est différente. Pour résumer : si le défilement doit être vers la gauche, le saut se réalise. S'il doit être vers la droite, le programme continue en séquence. Voyons ce dernier cas :

Ligne 11

Il nous faut d'abord calculer l'adresse de l'avant-dernier octet du premier trait de la ligne concernée, grâce à la formule : 49150 + (80 * NL).

Rappelons que HL est toujours chargé avec 80 * NL. La ligne 11 charge maintenant DE avec 49150.

Ligne 12

HL est additionné à DE, et le résultat se retrouve dans HL, qui est maintenant pointé correctement pour le transfert du premier trait. Nous allons maintenant pointer DE.

Lignes 13 et 14

L'empilement de HL, suivi immédiatement du dépilement de DE, a pour résultat que les deux registres sont maintenant chargés de la même manière.

Ligne 15

Incrémentation de DE, qui est maintenant, lui aussi, pointé correctement pour le transfert. Sur l'exemple du préambule, HL et DE seraient respectivement pointés sur 49230 et 49231.

Ligne 16

Le compteur de transfert (ou compteur de trait) est chargé avec 79. Tout est prêt pour le transfert répétitif, mais n'oublions pas qu'il faut d'abord sauvegarder le contenu du dernier octet du trait.

Ligne 17

Réalisation de cette sauvegarde : le contenu de l'emplacement mémoire adressé par DE est chargé dans A. C'est donc dans ce registre A que nous pourrons, quand nous en aurons besoin, récupérer cette valeur.

Ligne 18

Transfert répétitif de bloc avec décrémentation. Cette instruction a été suffisamment été développée dans le préambule, et nous n'y reviendrons pas. Résumons simplement la situation lorsque ce transfert est terminé (là encore, nous prendrons l'exemple du premier trait de la figure du préambule) :

HL est chargé avec 49151, DE avec 49152, BC avec 0. Le premier trait est décalé d'une position vers la droite, et le contenu initial de 49231 est dans A.

Ligne 19

Le contenu de A est chargé dans l'emplacement mémoire adressé par DE. Cela se passe bien sûr de commentaire.

Le travail sur ce premier trait est donc terminé, restent les sept autres. Mais il s'agit de vérifier, avant de recommencer l'opération pour le trait suivant, si les huit traits n'ont pas déjà été décalés, auquel cas le programme serait terminé.

Ligne 20

Récupération, sur la pile, de la valeur actuelle du compteur général, qui se retrouve dans BC (rappelons que cette valeur tient sur un seul octet, qu'elle avait été chargée *dans B* avant empilement, et que la valeur éventuellement contenue dans C nous est indifférente).

Ligne 21

Une nouvelle instruction, une des plus puissantes du Z 80 : décrémentation de B et saut relatif si non nul (ici le saut éventuel est de + 1, en ligne 23).

A la huitième décrémentation de B, donc quand les huit traits auront été traités le saut ne se fera pas et le programme continuera en ligne 22, qui provoquera le retour au BASIC. Si ce n'est pas le cas, le programme saute à la ligne 23, pour traiter le trait suivant :

Ligne 23

La nouvelle valeur du compteur général est sauvegardée.

Ligne 24

Pour nous occuper du trait suivant, il nous faut d'abord pointer HL et DE respectivement sur l'avant-dernier et le dernier octets de ce trait.

On se rend compte qu'il est possible de pointer HL en ajoutant tout simplement 2127 à sa valeur actuelle (si par exemple nous venons de finir le premier trait, 49151 + 2127 = 51278). Cela est bien entendu valable quel que soit le numéro de trait.

Cette ligne 24 charge DE avec 2127.

Ligne 25

Saut relatif de −21, en ligne 12. Là, HL est additionné à DE, puis DE est chargé comme HL, puis DE est incrémenté, etc. La boucle est donc bouclée, et les huit traits successifs seront traités de la même manière.

Les lignes 26 à 40, s'occupant du défilement vers la gauche, sont l'exact pendant des lignes 11 à 25, vous pouvez donc les étudier seul. Rappelons simplement que HL doit être cette fois pointé sur le deuxième octet du trait concerné, et DE sur le premier. D'autre part, le transfert répétitif se fera avec *incrémentation*.

Ce programme fonctionne indifféremment dans les trois modes. Si vous fournissez un numéro de ligne erroné, il se contente en principe de ne pas fonctionner et ne se détruit pas.

En vous basant sur cet exemple, vous pouvez bien entendu concevoir d'autres programmes du même genre : défilement de plusieurs lignes, défilements inverses et simultanés, défilements verticaux, etc.

En voici un exemple d'utilisation :

```
100 MODE 1
110 LOCATE 1,12 : PRINT ''VOICI UN DÉFILEMENT VERS LA
    DROITE...''
120 FOR I=1 TO 300 : CALL 43830,12,1 : NEXT I
130 CLS : LOCATE 1,12 : PRINT ''... EN VOICI UN VERS LA
    GAUCHE''
140 FOR I = 1 TO 300 : CALL 43830,12,0 : NEXT I
150 CLS : LOCATE 8, 12 : PRINT ''ET ENFIN, LES 2
    ENSEMBLES''
160 FOR I = 1 TO 14 : CALL 43830,12,0 : NEXT I : FOR I =
    1 TO 14 : CALL 43830,12,1 : NEXT I : GOTO 160
```

5. DESSIN D'UN CERCLE

Ce programme est certainement l'un des plus complexes de tous ceux que présente cet ouvrage, essentiellement parce que nous allons y manipuler des variables de deux types : entières ou à virgule flottante.

A cet égard, il est indispensable d'avoir lu au préalable l'Annexe V, ou tout au moins sa première partie (Les variables numériques).

Le programme débute en 43680, finit en 43877 (chiffre de vérification : 27097), n'est pas directement relogeable, et son format d'appel est le suivant :

CALL 43680,XC,YC,R,C,PV

- XC et YC sont les coordonnées du centre du cercle.
- R est la longueur du rayon (exprimée en pixels).
- C est la couleur.
- PV détermine si le cercle doit être plein ou vide (0 : vide, 1 : plein).

Le paramètre PV sera optionnel. S'il n'est pas indiqué, le cercle sera vide.

Par ailleurs, une protection sera établie, qui provoquera un retour immédiat au BASIC si l'instruction CALL n'est pas suivie de quatre ou cinq paramètres. Avec certains programmes, en effet, l'oubli d'un ou de plusieurs paramètres peut provoquer des catastrophes. Ce genre de protection permet d'éviter ces désagréments.

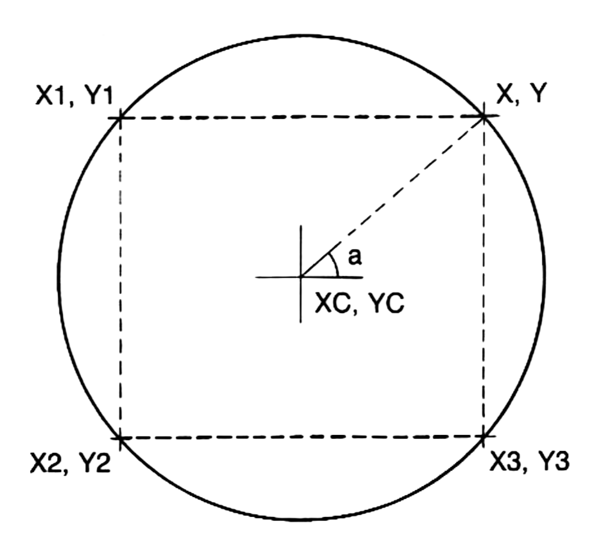
Vous savez certainement que l'équation d'un cercle se présente sous la forme :

$$X = XC + R * Cos(a)$$
$$Y = YC + R * Sin(a)$$

la valeur a étant la valeur de l'angle qui varie de 0 à 360 degrés.

Nous allons donc utiliser dans ce programme les routines &BD8B et &BD88 permettant de calculer respectivement les sinus et les cosinus. Ces routines sont malheureusement assez lentes et c'est pour gagner du temps que nous allons utiliser une petite astuce grâce à laquelle nous n'aurons à calculer que les sinus et cosinus des angles allant de 0 à 90 degrés au lieu d'avoir à le faire de 0 à 360 degrés.

La figure ci-dessous servira à expliquer le processus :



Chaque fois que nous aurons calculé les coordonnées d'un point (X,Y), dont l'angle correspondant variera de 0 à 90°, nous calculerons les coordonnées *relatives* par rapport à ce point du point (X1, Y1), puis les coordonnées relatives de (X2,Y2) par rapport à (X1,Y1), et enfin les coordonnées relatives par rapport à (X2,Y2) de (X3,Y3). Les quatre quadrants seront donc dessinés simultanément. Remarquons que les coordonnées relatives de :

(X1,Y1) par rapport à (X,Y) sont : − 2 (X − XC), 0 ;
(X2,Y2) par rapport à (X1,Y1) sont : 0, − 2 (Y − YC) ;
(X3,Y3) par rapport à (X2,Y2) sont : 2 (X − XC), 0.

A ce sujet, revoir éventuellement l'Annexe II.

Il faut enfin savoir que nous aurons, au cours de ce programme, à stocker provisoirement un certain nombre de variables. C'est la raison pour laquelle les cases mémoires 43878 à 43895 sont divisées en cinq "placards" dans lesquels nous rangerons ces variables (voir en fin de listing la manière dont sont disposés ces placards).

Voyons maintenant le déroulement ligne par ligne de ce programme (dans un premier temps, nous laisserons de côté les lignes 1 à 17, qui rendent le cinquième paramètre optionnel et installent la protection évoquée plus haut). Considérons donc pour l'instant que le programme débute en ligne 18, et qu'il n'y a que quatre paramètres derrière notre instruction CALL (XC,YC,R,C). Le cercle tracé sera donc vide.

LIGNE	ADRESSE DU 1ER CODE DE CETTE LIGNE DEC.	HEXA.	CODE-MACHINE	MNEMONIQUE
1	43680		21,30,AB	LD HL,43824
2	43683		36,ED	LD (HL),237
3	43685		21,41,AB	LD HL,43841
4	43688		36,ED	LD (HL),237
5	43690		FE,4	CP 4
6	43692		28,16	JR Z,22
7	43694		FE,5	CP 5
8	43696		C0	RET NZ
9	43697		AF	XOR A
10	43698		DD,BE,0	CP (IX+0)
11	43701		28,A	JR Z,10
12	43703		21,30,AB	LD HL,43824
13	43706		36,F9	LD (HL),249
14	43708		21,41,AB	LD HL,43841
15	43711		36,F9	LD (HL),249
16	43713		DD,23	INC IX
17	43715		DD,23	INC IX
18	43717		DD,22,75,AB	LD (43893),IX
19	43721		3E,5A	LD A,90
20	43723		32,77,AB	LD (43895),A
21	43726		DD,7E,0	LD A,(IX+0)
22	43729		CD,DE,BB	CALL 48094
23	43732		DD,66,3	LD H,(IX+3)
24	43735		DD,6E,2	LD L,(IX+2)
25	43738		11,66,AB	LD DE,43878
26	43741		CD,40,BD	CALL 48448
27	43744		2A,77,AB	LD HL,(43895)
28	43747		11,6B,AB	LD DE,43883
29	43750		CD,40,BD	CALL 48448
30	43753		CD,8B,BD	CALL 48523
31	43756		11,66,AB	LD DE,43878
32	43759		CD,61,BD	CALL 48481
33	43762		CD,46,BD	CALL 48454
34	43765		DD,2A,75,AB	LD IX,(43893)
35	43769		CD,58,AB	CALL 43864
36	43772		19	ADD HL,DE
37	43773		22,6B,AB	LD (43883),HL
38	43776		2A,77,AB	LD HL,(43895)
39	43779		11,70,AB	LD DE,43888
40	43782		CD,40,BD	CALL 48448
41	43785		CD,88,BD	CALL 48520
42	43788		11,66,AB	LD DE,43878
43	43791		CD,61,BD	CALL 48481
44	43794		CD,46,BD	CALL 48454
45	43797		DD,2A,75,AB	LD IX,(43893)
46	43801		CD,5F,AB	CALL 43871
47	43804		19	ADD HL,DE
48	43805		ED,5B,6B,AB	LD DE,(43883)
49	43809		E5	PUSH HL
50	43810		D5	PUSH DE
51	43811		CD,EA,BB	CALL 48106
52	43814		CD,58,AB	CALL 43864
53	43817		CD,4A,AB	CALL 43850
54	43820		C1	POP BC
55	43821		D5	PUSH DE

```
43822         C5          PUSH BC
43823         CD,ED,BB    CALL 48109
43826         CD,5F,AB    CALL 43871
43829         CD,4A,AB    CALL 43850
43832         EB          EX DE,HL
43833         CD,ED,BB    CALL 48109
43836         E1          POP HL
43837         CD,50,AB    CALL 43856
43840         CD,ED,BB    CALL 48109
43843         21,77,AB    LD HL,43895
43846         35          DEC (HL)
43847         F8          RET M
43848         18,96       JR - 106
43850  AB4A   C1          POP BC
43851         E1          POP HL
43852         C5          PUSH BC
43853         ED,52       SBC HL,DE
43855         29          ADD HL,HL
43856  AB50   CD,C7,BD    CALL 48583
43859         EB          EX DE,HL
43860         21,0,0      LD HL,0
43863         C9          RET
43864  AB58   DD,56,7     LD D,(IX+7)
43867         DD,5E,6     LD E,(IX+6)
43870         C9          RET
43871  AB5F   DD,56,5     LD D,(IX+5)
43874         DD,5E,4     LD E,(IX+4)
43877         C9
```

Le tableau suivant représente les cinq placards :

43878	AB66	Placard R (rayon)
43883	AB6B	Placard Cosinus/Cos * R/XC + Cos * R
43888	AB70	Placard Sinus/Sin * R/YC + Sin * R
43893	AB75	Placard IX
43895	AB77	Placard angle/compteur

Intéressons nous tout d'abord au bloc constitué par les lignes 18 à 37, qui se charge d'effectuer le calcul de l'abscisse X (rappelons que X = XC + R * Cos (a)).

Ligne 18

Le contenu du registre IX est rangé dans le placard prévu à cet effet (cases mémoire 43893 et 43894). Ce registre est chargé pour l'instant, comme nous le savons, avec l'adresse de l'octet faible de C, mais les routines Sinus et Cosinus que nous appellerons plus loin utilisent ce registre pour leur propre compte et vont donc dénaturer son contenu, c'est pourquoi il faut le sauvegarder.

Ligne 19

Chargement de l'accumulateur avec 90. Ce chiffre servira à la fois pour initialiser le compteur, et pour calculer les sinus et cosinus successifs des angles 90 à 0.

Ligne 20

Le contenu de A est rangé dans son placard (case mémoire 43895). Une seule case suffit car la machine "sait" que le contenu de A ne peut en aucun cas être supérieur à 255.

Lignes 21 et 22

Chargement de A avec le numéro de couleur choisi et appel de la routine &BBDE, qui fixe la couleur graphique.

Lignes 23 et 24

Chargement de HL avec R.

Ligne 25

Chargement de DE avec l'adresse de la première des cinq cases constituant le placard réservé au rayon R. Nous allons en effet transformer la représentation de R (qui est pour l'instant stockée sous une forme entière, c'est-à-dire sur 2 octets) en sa représentation en virgule flottante, donc sur 5 octets. Nous y sommes obligés car nous allons travailler avec le bloc de routines "Arithmétique avec virgule flottante".

Bien entendu, la valeur R reste la même, seule change sa représentation.

Ligne 26

Appel de la routine &BD40 : "Transformer la représentation entière d'un nombre en sa représentation en virgule flottante".

Le nombre traité doit au départ être chargé dans HL et l'adresse du placard où l'on veut qu'il se retrouve doit être chargée dans DE.

Après l'exécution de cette ligne, la représentation en virgule flottante de R est donc rangée dans les cases mémoire 43878 à 43882.

Ligne 27

Chargement de HL avec le contenu de 43895, c'est-à-dire avec la valeur de l'angle à ce moment : 90 au premier tour de boucle, 89 au deuxième, 88 au troisième, etc.

Ligne 28

Chargement de DE avec l'adresse du placard réservé aux cosinus (on peut également dire : pointage de DE sur ce placard).

Ligne 29

Comme pour le rayon, transformation en virgule flottante et rangement du résultat au placard.

Il faut savoir qu'au retour de la routine &BD40, les contenus des registres HL et DE ne sont plus les mêmes. Au cours de ses manipulations, la routine les modifie et, après son exécution, c'est maintenant HL, et non plus DE, qui est chargé avec l'adresse du placard.

La manière dont doit être chargé un registre avant l'appel d'une routine s'appelle le *registre d'appel*, et son état lorsqu'il revient le *registre de réponse*.

Ligne 30

Appel de la routine &BD8B qui calcule le cosinus d'un angle.

Registre d'appel : HL doit être chargé avec l'adresse de la variable que l'on veut traiter (ce qui était bien le cas après la ligne 29).

Registre de réponse : HL reste inchangé. Le résultat est rangé à la place de la variable.

Ligne 31

Chargement de l'adresse du rayon R dans DE, en prévision de l'utilisation de la routine &BD61 en ligne 32.

Ligne 32

Appel de la routine &BD61 qui effectue la multiplication de deux nombres.

Registres d'appel : HL et DE doivent respectivement être chargés avec l'adresse du premier et du deuxième nombre.

Registres de réponse : HL reste inchangé, mais pas DE. Le résultat de la multiplication est chargé à l'adresse pointée par HL (notons que le nombre adressé au départ par DE reste inchangé).

Ligne 33

Appel de la routine &BD45, qui effectue l'opération inverse de la routine &BD40, puisqu'elle transforme une représentation en virgule flottante en une représentation entière.

Registre d'appel : HL doit être chargé avec l'adresse du nombre.

Registre de réponse : HL contient le nombre entier, c'est-à-dire maintenant R * Cos(a).

Ligne 34

Chargement de IX avec le contenu de l'emplacement mémoire d'adresse 43893. Ce registre est donc maintenant rechargé avec sa valeur initiale sauvegardée en ligne 18.

Ligne 35

Une ligne un peu particulière : il s'agit de l'appel du sous-programme interne d'adresse 43864 (lignes 78 à 80). Ce sous-programme a pour fonction de charger DE avec XC. Nous aurons en effet besoin d'effectuer plusieurs fois cette opération, et il est donc rentable de faire un sous-programe spécifique qu'il suffit d'appeler. Ce sous-programme est extrêmement simple :

Lignes 78 et 79 : Chargement de DE avec XC

Ligne 80 : Retour de sous-programme (donc à la ligne 36)

Plusieurs remarques s'imposent néanmoins :

1. Cette routine permet d'économiser trois octets chaque fois que l'on aura besoin de charger DE avec XC. L'appel nécessite trois octets ; mais, s'il fallait charger directement DE, il en faudrait six.
2. L'instruction de retour d'un sous-programme interne est la même que celle d'un retour de l'instruction CALL.

3. La règle évoquée au début de l'ouvrage est plus que jamais valable : si la pile est manipulée au cours d'un sous-programme, *elle doit être remise en état* avant que le retour soit provoqué par l'instruction C9.

Ligne 36

DE étant maintenant chargé avec XC et HL avec R * Cos(a), ils sont additionnés pour obtenir enfin l'abscisse de notre premier point : XC + R * Cos(a).

Le résultat de cette addition se trouve dans HL.

Ligne 37

Mise au placard de ce résultat. Notons qu'il s'agit d'un entier, maintenant, et qu'il est donc rangé sur 2 octets : 43883 et 43884.

Les lignes que nous venons d'étudier peuvent en fait être séparées en blocs distincts :

- Le premier va des lignes 18 à 26 et se charge de sauvegarder IX, de sauvegarder la valeur du compteur, de transformer R en virgule flottante et de le ranger une fois pour toutes dans son placard. Ces différentes tâches n'auront à être exécutées qu'une seule fois.
- Le second bloc (lignes 27 à 36) se charge du calcul XC + R * Cos(a). La ligne 37 sauvegarde ce résultat.

Les lignes 38 à 47 sont identiques aux lignes 27 à 36, si ce n'est que la ligne 41 appelle la routine Sinus au lieu de Cosinus, et que l'on se sert dans ce bloc du placard réservé aux sinus. De même, l'appel au sous-programme 43871 de la ligne 46 charge DE avec YC au lieu de XC.

Il est donc inutile de refaire une étude ligne par ligne de ce bloc chargé de calculer YC + R * Sin(a), et nous nous contenterons de résumer la situation après son exécution (donc immédiatement après la ligne 47) :

1. L'abscisse X du point (X,Y) que nous voulons dessiner a été calculée précédemment et stockée à l'adresse 43883. Elle s'y trouve toujours.
2. L'ordonnée Y du même point vient d'être calculée et se trouve, après l'exécution de la ligne 47, dans HL.

Nous allons maintenant étudier le bloc allant de la ligne 48 à la ligne 64. Ce bloc va tracer le point (X,Y), puis (X1,Y1), puis (X2,Y2), et

enfin (X3,Y3). (Revoir à ce sujet la figure du début et les explications s'y rapportant.)

Ligne 48

Chargement de DE avec X.

Lignes 49 à 50

Sauvegarde, en les rangeant sur la pile, de Y puis de X (attention à la pile, ça va se compliquer...).

Ligne 51

Appel de la routine &BBEA qui dessine sur l'écran le point de coordonnées X,Y.

Registres d'appel : DE doit être chargé avec l'abscisse X et HL avec l'ordonnée Y.

Registres de retour : tous modifiés.

Ce premier point étant tracé, nous allons maintenant nous occuper du point (X1,Y1) dont les coordonnées relatives par rapport au précédent sont : – 2 (X – XC), 0. Nous devrons donc d'abord calculer – 2 (X – XC).

Ligne 52

Chargement de DE avec XC.

Ligne 53

Appel du sous-programme interne d'adresse 43850, allant des lignes 69 à 77.

Voyons ce que fait cette routine :

> Lignes 69, 70 et 71, ou "l'art et la manière d'éviter un énorme piège".
> En effet, qu'y a-t-il sur la pile au moment où l'on arrive à la ligne 69 ? X ? Eh bien non, justement. N'oubliez pas que, lorsque le programme rencontre un appel de sous-programme, l'adresse de retour est empilée et ne sera dépilée qu'au moment du retour. Au sommet de la pile se trouve donc pour l'instant l'adresse 43820.
> Cette adresse est soulevée, X qui est en-dessous est chargé dans HL, puis l'adresse est reposée sur la pile.
>
> Ligne 72 : DE est soustrait de HL. Le résultat (X – XC) se retrouve dans HL.

Ligne 73 : HL est additionné à lui-même, ce qui revient à le multiplier par 2. HL contient maintenant 2 (X – XC).

Ligne 74 : Appel de la routine de changement de signe déjà étudiée. HL est maintenant chargé avec – 2 (X – XC).

Ligne 75 : Échange des contenus de HL et DE.

Ligne 76 : HL est chargé avec 0.

Ligne 77 : Retour de sous-programme. Rappelons encore une fois que l'adresse de retour est en même temps dépilée.

Lignes 54, 55, 56

Sauvegarde de – 2 (X – XC), en s'arrangeant pour que Y reste au sommet de la pile.

Ligne 57

HL étant toujours chargé avec 0 et DE avec – 2 (X – XC), appel de la routine &BBED qui dessine un point à une position relative au curseur graphique, ce dernier se trouvant alors en (X,Y), c'est-à-dire au dernier point tracé.

Registres d'appel : l'abscisse et l'ordonnée relatives doivent se trouver respectivement dans DE et HL.

Notre deuxième point (X1,Y1) est donc tracé et le curseur graphique y est positionné. Nous pouvons nous attaquer au troisième point (X2,Y2) de coordonnées relatives 0, – 2 (Y – YC).

Ligne 58

Appel du sous-programme interne d'adresse 43871 qui charge DE avec YC.

Ligne 59

Appel du sous-programme interne d'adresse 43850, déjà étudié à la ligne 53. Cette fois, il va calculer – 2 (Y – YC). Au retour, HL est chargé avec 0 et DE avec – 2 (Y – YC).

Ligne 60

Inversion de HL et D. Cette fois en effet, c'est l'abscisse qui doit être égale à 0.

Ligne 61

Appel de la routine &BBED, déjà étudiée. Notre troisième point étant tracé, reste le quatrième, de coordonnées relatives 2 (X – XC), 0.

Ligne 62

Si vous avez bien suivi l'état de la pile, vous savez qu'il s'y trouve actuellement au sommet la valeur – 2 (X – XC), sauvegardée en ligne 55.

La ligne 62 charge cette valeur dans HL.

Ligne 63

Appel du sous-programe interne d'adresse 43856 (ligne 74). Notons que ce programme est en partie le même que celui qui est appelé aux lignes 53 et 59.

Le signe de HL est inversé, HL et DE sont échangés et HL est chargé avec 0. Au retour de cette routine, HL est donc chargé avec 0 et DE avec 2 (X – XC).

Ligne 64

Tracé du quatrième point (X3,Y3).

Nos quatre points étant tracés, nous allons passer à l'angle suivant après avoir vérifié l'état du compteur. C'est le rôle des lignes 65 à 67.

Ligne 65

HL est chargé avec l'adresse du placard Angle.

Ligne 66

Une nouvelle instruction : "Décrémenter le contenu de l'emplacement mémoire adressé par HL." L'angle est donc diminué de 1.

Ligne 67

Encore une nouvelle instruction : "Retour conditionnel de sous-programme." Si le résultat de l'opération de la ligne 66 est inférieur à 0, alors le retour au BASIC est effectué, sinon le programme se poursuit en séquence.

Ligne 68

Saut relatif de –106 (en ligne 27). Le programme va ainsi boucler jusqu'à ce que toutes les valeurs de l'angle, de 90 à 0, aient été traitées.

Les trois blocs suivants sont les trois sous-programmes déjà étudiés :

Lignes 69 à 77

Calcul de −2(X − XC) ou −2(Y − YC).

Lignes 78 à 80

Chargement de DE avec XC.

Lignes 81 à 83

Chargement de DE avec YC.

Nous pouvons maintenant étudier les lignes 1 à 17 qui rendent optionnel le paramètre PV (plein / vide) et protègent le programme en provoquant un retour immédiat au BASIC s'il n'y a pas quatre ou cinq paramètres.

Détaillons tout d'abord la procédure qui va nous permettre de dessiner un cercle plein.

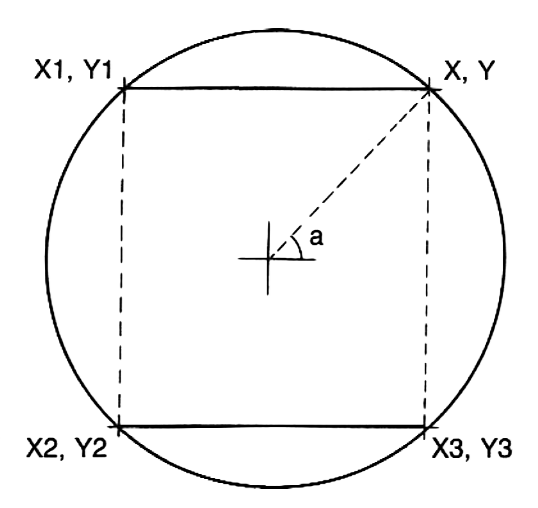
Reconsidérons la figure du début. Le processus de tracé des quatre points doit être maintenant bien clair pour tout le monde :

1. Tracé du point (X,Y).
2. Tracé du point (X1,Y1) relativement à X,Y.
3. Tracé du point (X2,Y2) relativement à (X1,Y1).
4. Tracé du point (X3,Y3) relativement à (X2,Y2).

Mais que se passerait-il si l'on y apportait les modifications suivantes :

1. Tracé du point (X,Y).
2. Tracé d'une ligne relative d'extrémité (X1,Y1).
3. Tracé du point (X2,Y2) relativement à (X1,Y1).
4. Tracé d'une ligne relative d'extrémité (X3,Y3).

Eh bien, tout simplement ceci :



En procédant ainsi, les lignes tracées les unes contre les autres, au fur et à mesure que a varie de 90 à 0, formeront un cercle plein.

Le point (X1,Y1) est tracé en ligne 57 grâce à la routine &BBED et le point (X3,Y3) en ligne 64 grâce à la même routine. Il suffirait donc de remplacer cette routine &BBED par la routine &BBF9 qui trace une ligne de la position du curseur jusqu'à la position relative indiquée.

Les lignes 57 et 64 se situent en mémoire comme ceci :

43823 : CD	43840 : CD
43824 : ED	43841 : ED
43825 : BB	43842 : BB

On peut donc se contenter de charger les cases mémoire 43824 et 43841 avec &F9 pour obtenir un appel non plus à &BBED, mais à &BBF9.

C'est ce que réalise le bloc de lignes 5 à 17 :

Ligne 5

Cette ligne nous permet une découverte extrêmement importante : il faut savoir en effet qu'au moment de l'appel CALL le nombre de paramètres qui se trouvent effectivement derrière l'instruction est automatiquement chargé dans l'accumulateur.

Cette ligne 5 effectue une comparaison (déjà étudiée) entre l'accumulateur et le chiffre 4. Si le résultat est 0, cela voudra donc dire qu'il y a exactement quatre paramètres derrière l'instruction CALL.

Ligne 6

Si le résultat de l'opération de la ligne 5 est 0, alors saut relatif de 22. Cela signifie en effet que le paramètre PV est omis, et que le cercle doit être vide.

Si ce n'est pas le cas, le programme se poursuit en séquence.

Ligne 7

Comparaison entre l'accumulateur et 5.

Ici se trouve la protection évoquée plus haut : à ce point du programme, nous savons déjà qu'il n'y a pas quatre paramètres. Nous vérifions maintenant s'il y en a cinq. Si c'est le cas, le résultat de l'opération sera 0, sinon il sera différent de 0.

Ligne 8

S'il n'y a ni quatre ni cinq paramètres, c'est donc qu'il y a une erreur dans le format d'appel. Dans ce cas, cette instruction "Retour de sous-programme si non nul" provoque le retour au BASIC.

Dans le cas contraire, c'est que le paramètre PV a été mis, et qu'il va donc falloir le tester.

Ligne 9

Littéralement : "OU exclusif logique entre l'accumulateur et lui-même." Sans entrer dans les détails, il suffit de savoir que cette instruction nous sert à mettre l'accumulateur à 0 (c'est plus simple que d'utiliser l'instruction de chargement).

Ligne 10

Comparaison entre A (chargé avec 0) et le paramètre PV.

Ligne 11

Si le résultat de l'opération de la ligne 10 est 0, alors le cercle doit être vide et cette ligne provoque un saut relatif de 10 (en ligne 16). Sinon, le cercle doit être plein et le programme se poursuit en séquence. Voyons d'abord ce dernier cas.

Ligne 12

Chargement de HL avec l'adresse 43824, celle qui doit être modifiée de manière à obtenir l'appel de la routine traçant une ligne au lieu d'un point.

Ligne 13

Chargement de la case mémoire adressée par HL avec la donnée 249 (&F9).

Lignes 14 et 15

Même processus, mais avec l'adresse 43841.

Lignes 16 et 17

Deux incrémentations successives de IX ; voyons pourquoi :
Nous avons dit que la partie principale du programme a été écrite en posant comme hypothèse que l'instruction CALL ne serait suivie que de quatre paramètres (il fallait bien choisir entre quatre et cinq puisque PV est optionnel). Cette hypothèse implique bien entendu que le registre IX est pointé au départ comme le montre la partie gauche du tableau ci-après :

	Pointage de IX si 4 Paramètres	Pointage de IX si 5 Paramètres
PV		IX + 0
		IX + 1
C	IX + 0	IX + 2
	IX + 1	IX + 3
R	IX + 2	IX + 4
	IX + 3	IX + 5
YC	IX + 4	IX + 6
	IX + 5	IX + 7
XC	IX + 6	IX + 8
	IX + 7	IX + 9

Si l'instruction CALL est suivie de cinq paramètres, par contre le pointage de IX est tel que montré dans la colonne de droite.

Les pointages sont donc différents selon le cas, ce qui pose problème puisque le programme est commun.

La double décrémentation des lignes 16 et 17 apporte la solution en rendant ces pointages identiques. Ainsi, si nous utilisons le programme en transmettant cinq paramètres et que par exemple YC doive être chargé, nous ne serons plus obligés d'utiliser IX + 6 et IX + 7, puisque la valeur 2 aura été au préalable ajoutée à IX. Nous utiliserons donc

IX + (6 — 2) et IX + (7 — 2), soit IX + 4 et IX + 5, comme dans le cas où il y a quatre paramètres.

Le saut relatif de la ligne 11 s'explique de la même manière.

Revenons maintenant aux lignes 1 à 4.

Elles initialisent le programme en chargeant les cases mémoire 43824 et 43841 avec la donnée 237 (&ED). Après ces lignes, le programme est donc un programme "cercle vide". Pourquoi ces lignes ?

Imaginons qu'elles n'existent pas et que dans un programme BASIC, vous ayez à utiliser le programme cercle plein. Les lignes 12 à 15 vont faire les modifications nécessaires et cela marchera. Mais si jamais, dans le même programme BASIC, vous avez à utiliser ensuite le programme "Cercle vide", cela ne marchera plus, et pour cause : si la modification vide → plein est prévue, il n'en est pas de même pour l'inverse plein → vide et vous ne pourrez plus dessiner que des cercles pleins. C'est pourquoi les lignes 1 à 4 initialisent systématiquement le programme "Cercle vide" avant le test éventuel du paramètre PV.

Remarque

Revenons un court instant aux lignes 20 et 65. La première charge l'emplacement mémoire 43895 avec 90 (il s'agit de l'angle de départ). La ligne 65 charge le registre HL avec le contenu de l'emplacement mémoire d'adresse 43895.

Mais attention : comme il s'agit d'un registre double, L sera chargé avec (43895), *et H avec (43896).* Or, ce dernier emplacement n'est en rien modifié par la ligne 20, puisque le chargement se fait alors grâce à un registre simple. Si donc, au moment ou le programme a été chargé, il n'y avait rien (donc 0) à cet emplacement, tout ira bien et HL sera correctement chargé avec la valeur de l'angle. Mais si un programme est déjà venu occuper cette adresse, il se peut que l'emplacement 43896 soit toujours chargé avec une valeur quelconque. Dans ce cas, et pour éviter que HL soit chargé avec une valeur incorrecte, il faudra insérer, avant ou après le programme de chargement, la ligne :

```
POKE 43896,0
```

Naturellement, cette précaution pourrait aussi être facilement intégrée dans le programme lui-même.

L'étude de ce programme étant terminée, voici quelques exemples de son utilisation :

```
100 CLS:DEG
110 FOR I=110 TO 10 STEP -11
120 C=C+1: IF C=4 THEN C=1
130 CALL 43680,200,200,I,C : CALL 43680,400,200,I,C,1
140 NEXT I
```

Évidemment, une fois un programme en langage machine au point, on peut toujours s'amuser à le "torturer" un peu.

Essayez par exemple de supprimer le DEG de la ligne 100.

Ou bien laissez le DEG mais insérez la ligne suivante :

```
105 POKE 43722,50
```

Ou encore, enlevez le DEG et insérez :

```
105 POKE 43722,5
```

Un dernier exemple enfin, encore plus spectaculaire. Insérez la ligne :

```
105 POKE 43722,5 : POKE 43834,&F9
```

et modifiez le "I=110" de la ligne 110 en "I=165". Essayez une fois avec lé DEG et une fois sans.

Il reste à signaler que le programme fonctionne indifféremment dans les trois modes, et à parler un peu de sa vitesse d'exécution.

Pour un cercle vide, un programme BASIC aura un temps d'exécution d'environ 12,8 secondes, alors que notre programme en langage machine en aura un d'environ 2,8 secondes. Le gain de temps n'est certes pas négligeable, mais on ne peut pas dire non plus que ce soit extraordinaire.

Cette relative lenteur est due, comme nous l'avons déjà dit, aux routines Sinus et Cosinus. La difficulté est incontournable, tout au moins d'une manière parfaitement élégante, à moins de fabriquer de toutes pièces un algorithme sinus et cosinus, ce qui est un autre problème.

Néanmoins, pour les "mordus" du cercle, le programme suivant propose une autre solution et exécute la même tâche en un peu plus d'une demi-seconde. Mais on n'a rien sans rien, et cela va nous coûter quelque 910 octets, non pas à cause du programme lui-même, rassurez-vous, mais tout simplement parce que nous allons avoir besoin d'un énorme placard !

6. CERCLE RAPIDE

Attention, il est absolument nécessaire d'avoir au préalable étudié le programme précédent avant de passer à celui-ci.

Ce programme de cercle rapide est basé sur une idée fort simple : puisque les routines Sinus et Cosinus sont lentes, nous allons nous arranger pour les utiliser le moins possible. Pour ce faire, une solution vient immédiatement à l'esprit : calculer les sinus et cosinus des angles qui nous intéressent une fois pour toutes, et les ranger dans un tableau où nous n'aurons plus qu'à les lire.

Ce tableau comprendra 910 octets (de 42985 à 43894), et son remplissage se fera grâce à un petit programme BASIC qu'il suffira de lancer une fois avant le premier appel du programme "Cercle rapide". On pourra ensuite utiliser ce dernier autant de fois que l'on voudra, et la relative longueur d'exécution du remplissage (environ 8,6 secondes) n'est de ce fait pas gênante.

Voici le programme BASIC :

```
10 MEMORY 42984 : DEG : T=42985
20 FOR I=90 TO 0 STEP -1
30 A=COS(I) : FOR H=0 TO 4 : POKE T+H,PEEK(@ A+H) :
   NEXT H : T=T+5
40 A=SIN(I) : FOR H=0 TO 4 : POKE T+H,PEEK(@ A+H) :
   NEXT H : T=T+5
50 NEXT I
```

Si vous avez lu en Annexe V la section concernant les variables numériques, vous devriez le comprendre facilement :

La représentation en virgule flottante de COS(90) est stockée sur 5 octets (42985 à 42989), puis celle de SIN(90) sur les cinq suivants (42990 à 42994), puis celle de COS(89) sur les cinq suivants, puis celle de SIN(89), et ainsi de suite jusqu'à 0 compris. En bref, les cosinus et sinus alternent dans le tableau de 5 en 5 octets.

La plus grande partie du programme fonctionne exactement de la même manière que le précédent et beaucoup de lignes sont identiques. Elles ne seront plus expliquées autrement que par un renvoi aux lignes ou blocs de lignes identiques du programme précédent.

Il ne faut toutefois pas oublier que, ce programme n'ayant pas la même situation en mémoire, les adresses internes (placards et sous-

programmes internes) seront différentes. Les placards, par exemple, commencent à l'adresse 42967 (&A7D7). Cela n'empêche évidemment pas que les démarches soient identiques.

Précisons enfin que toute la première partie (rendant PV optionnel et installant la protection) n'a pas été réécrite. Vous pouvez, bien sûr, l'ajouter.

Tel qu'il est donc, ce programme ne peut soutenir que quatre paramètres et ne dessine que des cercles vides. Il commence en 42800, finit en 42966, n'est pas directement relogeable, et son format d'appel est le suivant :

CALL 42800,XC,YC,R,C

Son chiffre de vérification est 26406.

Une petite astuce : si vous ne vous sentez pas le courage nécessaire pour ajouter la partie permettant l'option plein / vide, essayez ceci :

- Pour un cercle plein, ajoutez, immédiatement avant l'appel du programme, la ligne :

POKE 42927,&F9 : POKE 42910, &F9

- Pour un cercle vide, remplacez &F9 par &ED.

Ce procédé est, disons-le, assez rudimentaire et plutôt lourd à utiliser, mais il a au moins le mérite d'être simple à mettre en place.

Voici le listing de ce sixième programme :

LIGNE	ADRESSE DU 1ER CODE DE CETTE LIGNE DEC.	HEXA.	CODE-MACHINE	MNEMONIQUE
1	42800		DD,22,E6,A7	LD (42982),IX
2	42804		3E,5A	LD A,90
3	42806		32,E8,A7	LD (42984),A
4	42809		DD,7E,0	LD A,(IX+0)
5	42812		CD,DE,BB	CALL 48094
6	42815		DD,66,3	LD H,(IX+3)
7	42818		DD,6E,2	LD L,(IX+2)
8	42821		11,D7,A7	LD DE,42967
9	42824		CD,40,BD	CALL 48448
10	42827		21,E9,A7	LD HL,42985
11	42830		1,5,0	LD BC,5
12	42833		C5	PUSH BC
13	42834		11,DC,A7	LD DE,42972
14	42837		ED,B0	LDIR
15	42839		D5	PUSH DE
16	42840		E5	PUSH HL

42841		21,DC,A7	LD HL,42972
42844		11,D7,A7	LD DE,42967
42847		CD,61,BD	CALL 48481
42850		CD,46,BD	CALL 48454
42853		DD,2A,E6,A7	LD IX,(42982)
42857		CD,C9,A7	CALL 42953
42860		19	ADD DE,HL
42861		22,DC,A7	LD (42972),HL
42864		E1	POP HL
42865		D1	POP DE
42866		C1	POP BC
42867		C5	PUSH BC
42868		ED,B0	LDIR
42870		E5	PUSH HL
42871		21,E1,A7	LD HL,42977
42874		11,D7,A7	LD DE,42967
42877		CD,61,BD	CALL 48481
42880		CD,46,BD	CALL 48454
42883		DD,2A,E6,A7	LD IX,(42982)
42887		CD,D0,A7	CALL 42960
42890		19	ADD DE,HL
42891		ED,5B,DC,A7	LD DE,(42972)
42895		E5	PUSH HL
42896		D5	PUSH DE
42897		CD,EA,BB	CALL 48106
42900		CD,C9,A7	CALL 42953
42903		CD,BB,A7	CALL 42939
42906		C1	POP BC
42907		D5	PUSH DE
42908		C5	PUSH BC
42909		CD,ED,BB	CALL 48109
42912		CD,D0,A7	CALL 42960
42915		CD,BB,A7	CALL 42939
42918		EB	EX HL,DE
42919		CD,ED,BB	CALL 48109
42922		E1	POP HL
42923		CD,C1,A7	CALL 42945
42926		CD,ED,BB	CALL 48109
42929		D1	POP DE
42930		C1	POP BC
42931		21,E8, A7	LD HL,42985
42934		35	DEC (HL)
42935		F8	RET M
42936		EB	EX HL,DE
42937		18,96	JR -106
42939	A7BB	C1	POP BC
42940		E1	POP HL
42941		C5	PUSH BC
42942		ED,52	SBC HL,DE
42944		29	ADD HL,HL
42945	A7C1	CD,C7,BD	CALL 48583
42948		EB	EX HL,DE
42949		21,0,0	LD HL,0
42952		C9	RET
42953	A7C9	DD,56,7	LD D,(IX+7)
42956		DD,5E,6	LD E,(IX+6)
42959		C9	RET
42960	A7D0	DD,56,5	LD D,(IX+5)
42963		DD,5E,4	LD E,(IX+4)
42966		C9	RET

Lignes 1 à 9

Même processus et même fonction que les lignes 18 à 26 du programme précédent.

Le bloc de lignes 10 à 14 va maintenant se charger de recopier la représentation de Cos(a), stockée dans le tableau sur 5 octets, dans le placard Cosinus.

Ligne 10

HL est pointé sur le début du tableau.

Ligne 11

BC est chargé avec la valeur 5. Ce sera le compteur de transfert.

Ligne 12

Sauvegarde de cette valeur que nous aurons à réutiliser.

Ligne 13

DE est pointé sur le placard Cosinus.

Ligne 14

Transfert répétitif de bloc avec incrémentation, déjà étudié dans le programme "Défilement d'une ligne". Le contenu de l'octet sur lequel était pointé HL ainsi que les quatre suivants sont respectivement chargés dans l'octet sur lequel était pointé DE, et dans les quatre suivants. La valeur Cos(a) est donc bien maintenant stockée dans son placard (au premier tour de boucle, a=90).

A chaque tour de boucle, HL va ainsi parcourir le tableau de 5 en 5, pointant successivement sur Cos(a), Sin(a), cos(a−1), Sin(a−1), etc.

Après cette ligne 14, DE, qui a été lui aussi incrémenté cinq fois, est pointé sur le placard Sinus.

Lignes 15 et 16

Sauvegarde par empilement de DE, puis de HL.

Ligne 17

Pointage de HL sur le placard Cosinus.

Lignes 18 à 24

Même processus et même fonction que les lignes 31 à 37 du programme précédent : calcul et rangement de X (=XC+R * Cos(a)).

Lignes 25 à 29

Elles effectuent la même opération que les lignes 10 à 14 vues précédemment, mais cette fois, c'est le sinus qu'il s'agit de recopier dans son placard. Une autre différence : c'est par dépilement que les trois registres sont pointés ou initialisés. A la fin de ces lignes, HL se retrouve pointé, dans le tableau, sur le cosinus de l'angle suivant.

Ligne 30

Sauvegarde de HL.

Ligne 31

Avant de passer au bloc suivant, pointage de HL sur le placard sinus.

Lignes 32 à 54

Même processus et même fonction que les lignes 42 à 64 du programme précédent : calcul de Y (=YC+R * Cos(a)), récupération de X dans DE et tracé des quatre points.

Le bloc suivant (lignes 55 à 61) représente le compteur, qui est un peu différent de celui du programme précédent :

Lignes 55 et 56

Juste avant que ces deux lignes ne soient exécutées, l'état de la pile est le suivant :

- Au sommet se trouve l'adresse, dans le tableau, du cosinus de l'angle suivant.
- En dessous se trouve la valeur 5.
- Et encore en dessous l'adresse du retour au programme BASIC.

La pile est remise en état grâce à ces deux lignes, cela dans l'éventualité d'un retour au BASIC si le compteur indique que le programme est terminé. Dans le cas contraire, les deux valeurs qui se trouvaient sur la pile pourront toujours être récupérées dans DE et BC.

Ligne 57

HL est pointé sur le placard de l'angle (qu'il est peut-être abusif d'appeler "angle", puisqu'il ne nous sert que de compteur et non pas pour le calcul des sinus et cosinus).

Ligne 58

Décrémentation du contenu de l'emplacement mémoire adressé par HL (donc du compteur).

Ligne 59

Retour au BASIC si négatif (la boucle doit tourner 91 fois).

Ligne 60

Échange de HL et DE. Si la boucle n'est pas terminée, nous remettons dans HL l'adresse dans le tableau du cosinus de l'angle suivant. Cela étant fait et BC étant correctement chargé avec 5, nous pouvons effectuer un saut en début de boucle.

Ligne 61

Saut relatif de –106 (ligne 12) : la boucle est bouclée !

Lignes 62 à 70

Il s'agit du sous-programme déjà étudié précédemment et chargé de calculer –2(X–XC) ou –2(Y–YC).

Lignes 71 à 73

Sous-programme "Charger DE avec XC".

Lignes 74 à 76

Sous-programme "Charger DE avec YC".

L'étude de ce programme se termine là. Voici un exemple de son utilisation :

```
100 MODE 0 : DEG.
110 A=INT(RND × 640) : B=INT(RND × 400) : C=INT(RND
    × 110) : D=INT(RND × 16)
120 AL=INT(RND × 2) : IF AL=0 THEN E=&F9 ELSE E=&ED
130 POKE 42927,E : POKE 42910,E
140 CALL 42800,A,B,C,D
150 GOTO 110
```

Enfin (une petite facétie) essayez ceci :

```
100 MODE 1 : DEG
110 POKE 42927,&F9 : CALL 42800,320,200,100,3
```

7. DÉPLACEMENT D'UN MOBILE (4 DIRECTIONS)

Ceux d'entre vous qui s'y sont déjà essayé ont certainement pu se rendre compte des problèmes que pose la réalisation d'animations graphiques en BASIC. Dans beaucoup de cas, elles sont même impossibles à réaliser sans lenteurs excessives et phénomènes optiques désagréables, en particulier si le mobile couvre une surface d'écran importante.

Ce septième progamme est donc particulièrement intéressant puisqu'il permet le déplacement dans les quatre directions, avec une excellente qualité de mouvement, d'un objet couvrant une surface de 64 octets (pour comparaison, un caractère en mode 1 est défini sur 16 octets).

Le programme peut facilement être adapté pour un objet plus grand ou pour effectuer des déplacements dans d'autres directions (huit directions, par exemple).

Il est le plus long de tous ceux que présente cet ouvrage, et je vous conseillerai d'une part de procéder "lentement mais sûrement", et d'autre part de relire rapidement l'Annexe III traitant de la mémoire écran.

Il débute à l'adresse 43650, finit en 43897 (chiffre de vérification : 28800), n'est pas directement relogeable et son format d'appel est le suivant :

CALL 43650,D

Le paramètre D détermine la direction du mouvement :

1 → haut
2 → droite
3 → bas
4 → gauche

Ensuite, nous aurons besoin de deux placards de 2 octets chacun, pour y stocker deux valeurs que nous nommerons Y et Y1. Les adresses de ces placards sont respectivement 43898 (&AB7A) et 43900 (&AB7C).

Le programme peut être divisé en neuf blocs dont les rôles sont les suivants :

Bloc 1

(lignes 2 à 9) : Il réalise les branchements à telle ou telle partie du programme en fonction du sens de déplacement.

Bloc 2

(lignes 10 à 26) : Déplacement vers le bas.

Bloc 3

(lignes 27 à 43) : Déplacement vers le haut.

Bloc 4

(lignes 44 à 68) : Déplacement vers la droite.

Bloc 5

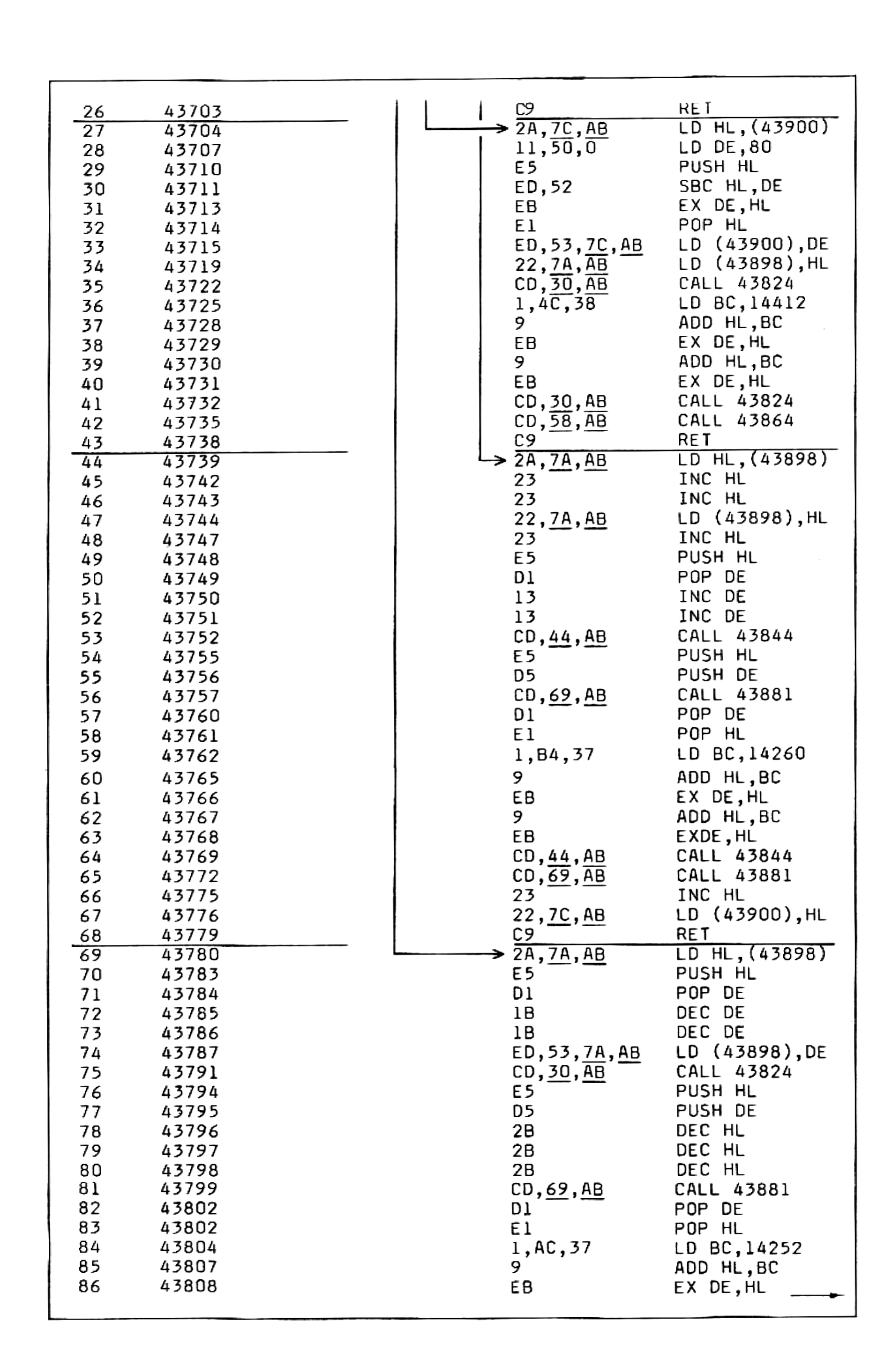
(lignes 69 à 94) : Déplacement vers la gauche.

Blocs 6, 7, 8, 9

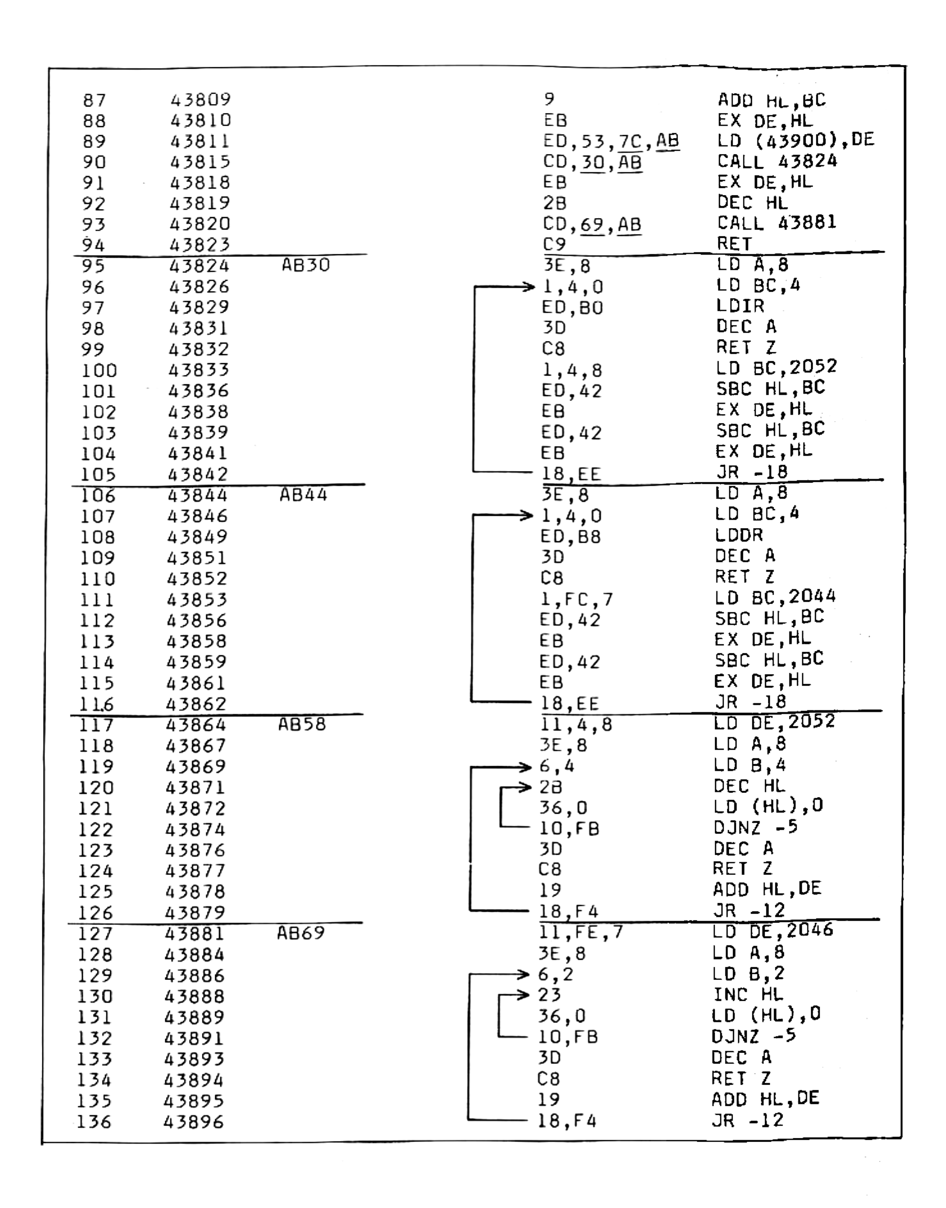
(respectivement lignes 95 à 105, 106 à 116, 117 à 126, 127 à 136) : Il s'agit là de quatre sous-programmes internes qui seront expliqués plus loin.

Voici le listing de ce programme :

LIGNE	ADRESSE DU 1erCODE DE CETTE LIGNE		CODE-MACHINE	MNEMONIQUE
	DEC.	HEXA.		
1	43650		CD,19,BD	CALL 48409
2	43653		DD,46,0	LD B,(IX+0)
3	43656		CB,50	BIT 2,B
4	43658		20,78	JRNZ,120
5	43660		CB,48	BIT 1,B
6	43662		28,28	JRZ,40
7	43664		CB,40	BIT 0,B
8	43666		20,2	JRNZ,2
9	43668		18,45	JR,68
10	43670		2A,7A,AB	LD HL,(43898)
11	43673		11,50,0	LD DE,80
12	43676		E5	PUSH HL
13	43677		19	ADD DE,HL
14	43678		EB	EX DE,HL
15	43679		E1	POP HL
16	43680		ED,53,7A,AB	LD (43898),DE
17	43684		22,7C,AB	LD (43900),HL
18	43687		CD,30,AB	CALL 43824
19	43690		1,AC37	LD BC,14252
20	43693		9	ADD HL,BC
21	43694		EB	EX DE,HL
22	43695		9	ADD HL,BC
23	43696		EB	EX DE,HL
24	43697		CD,30,AB	CALL 43824
25	43700		CD,58,AB	CALL 43864



26	43703	C9	RET
27	43704	2A,7C,AB	LD HL,(43900)
28	43707	11,50,0	LD DE,80
29	43710	E5	PUSH HL
30	43711	ED,52	SBC HL,DE
31	43713	EB	EX DE,HL
32	43714	E1	POP HL
33	43715	ED,53,7C,AB	LD (43900),DE
34	43719	22,7A,AB	LD (43898),HL
35	43722	CD,30,AB	CALL 43824
36	43725	1,4C,38	LD BC,14412
37	43728	9	ADD HL,BC
38	43729	EB	EX DE,HL
39	43730	9	ADD HL,BC
40	43731	EB	EX DE,HL
41	43732	CD,30,AB	CALL 43824
42	43735	CD,58,AB	CALL 43864
43	43738	C9	RET
44	43739	2A,7A,AB	LD HL,(43898)
45	43742	23	INC HL
46	43743	23	INC HL
47	43744	22,7A,AB	LD (43898),HL
48	43747	23	INC HL
49	43748	E5	PUSH HL
50	43749	D1	POP DE
51	43750	13	INC DE
52	43751	13	INC DE
53	43752	CD,44,AB	CALL 43844
54	43755	E5	PUSH HL
55	43756	D5	PUSH DE
56	43757	CD,69,AB	CALL 43881
57	43760	D1	POP DE
58	43761	E1	POP HL
59	43762	1,B4,37	LD BC,14260
60	43765	9	ADD HL,BC
61	43766	EB	EX DE,HL
62	43767	9	ADD HL,BC
63	43768	EB	EXDE,HL
64	43769	CD,44,AB	CALL 43844
65	43772	CD,69,AB	CALL 43881
66	43775	23	INC HL
67	43776	22,7C,AB	LD (43900),HL
68	43779	C9	RET
69	43780	2A,7A,AB	LD HL,(43898)
70	43783	E5	PUSH HL
71	43784	D1	POP DE
72	43785	1B	DEC DE
73	43786	1B	DEC DE
74	43787	ED,53,7A,AB	LD (43898),DE
75	43791	CD,30,AB	CALL 43824
76	43794	E5	PUSH HL
77	43795	D5	PUSH DE
78	43796	2B	DEC HL
79	43797	2B	DEC HL
80	43798	2B	DEC HL
81	43799	CD,69,AB	CALL 43881
82	43802	D1	POP DE
83	43802	E1	POP HL
84	43804	1,AC,37	LD BC,14252
85	43807	9	ADD HL,BC
86	43808	EB	EX DE,HL



87	43809		9	ADD HL,BC
88	43810		EB	EX DE,HL
89	43811		ED,53,7C,AB	LD (43900),DE
90	43815		CD,30,AB	CALL 43824
91	43818		EB	EX DE,HL
92	43819		2B	DEC HL
93	43820		CD,69,AB	CALL 43881
94	43823		C9	RET
95	43824	AB30	3E,8	LD A,8
96	43826		1,4,0	LD BC,4
97	43829		ED,B0	LDIR
98	43831		3D	DEC A
99	43832		C8	RET Z
100	43833		1,4,8	LD BC,2052
101	43836		ED,42	SBC HL,BC
102	43838		EB	EX DE,HL
103	43839		ED,42	SBC HL,BC
104	43841		EB	EX DE,HL
105	43842		18,EE	JR -18
106	43844	AB44	3E,8	LD A,8
107	43846		1,4,0	LD BC,4
108	43849		ED,B8	LDDR
109	43851		3D	DEC A
110	43852		C8	RET Z
111	43853		1,FC,7	LD BC,2044
112	43856		ED,42	SBC HL,BC
113	43858		EB	EX DE,HL
114	43859		ED,42	SBC HL,BC
115	43861		EB	EX DE,HL
116	43862		18,EE	JR -18
117	43864	AB58	11,4,8	LD DE,2052
118	43867		3E,8	LD A,8
119	43869		6,4	LD B,4
120	43871		2B	DEC HL
121	43872		36,0	LD (HL),0
122	43874		10,FB	DJNZ -5
123	43876		3D	DEC A
124	43877		C8	RET Z
125	43878		19	ADD HL,DE
126	43879		18,F4	JR -12
127	43881	AB69	11,FE,7	LD DE,2046
128	43884		3E,8	LD A,8
129	43886		6,2	LD B,2
130	43888		23	INC HL
131	43889		36,0	LD (HL),0
132	43891		10,FB	DJNZ -5
133	43893		3D	DEC A
134	43894		C8	RET Z
135	43895		19	ADD HL,DE
136	43896		18,F4	JR -12

La ligne 1 se passe de commentaires puisqu'il s'agit de l'appel de la routine de synchronisation avec le rayon (déjà étudiée). Nous commencerons donc par le bloc 1, mais auparavant, lancez sur votre machine le petit programme suivant :

```
10 FOR I=1 TO 4 : PRINT I,BIN$(I,8) : PRINT : NEXT
```

Ce qui apparaît sur l'écran après lancement est la représentation en code binaire des nombres 1, 2, 3 et 4 :

		7	6	5	4	3	2	1	0	←— Numéro du bit
1	—→	0	0	0	0	0	0	0	1	
2	—→	0	0	0	0	0	0	1	0	
3	—→	0	0	0	0	0	0	1	1	
4	—→	0	0	0	0	0	1	0	1	

Rassurez-vous, il ne nous est pas indispensable de comprendre la théorie de l'arithmétique binaire. Il suffit de savoir que la représentation en binaire d'un nombre compris entre 0 et 255 s'effectue sur ce que l'on appelle des bits, numérotés de 0 à 7 en partant de la droite et qui n'ont que deux états possibles : 0 ou 1.

Notons que si l'on se contente d'écrire dans le programme "BIN$(I)" au lieu de "BIN$(I,8)", la machine ne prend pas la peine de noter les 0 situés à gauche du dernier 1 (BIN$(4) donnerait par exemple 101). C'est pourquoi nous ajoutons le 8 qui force la machine à écrire les 8 bits.

Pour déterminer la valeur du paramètre D, nous pouvons donc raisonner comme suit :

- Si le bit n°2 de D est à 1, c'est que D vaut 4, et le programme doit sauter au bloc "Déplacement vers la gauche". Dans le cas contraire :

- Si le bit n°1 est mis à 0, il y a deux possibilités : D vaut 4 ou 1. Mais comme nous avons vu plus haut que D ne valait pas 4, D vaut donc 1 et le programme doit sauter au bloc "Déplacement vers le haut". Dans le cas contraire :

- Si le bit 0 de D est à 1, D peut valoir 1 ou 3. Nous avons vu précédemment que D ne valait pas 1, D vaut donc 3 et le programme doit sauter au bloc "Déplacement vers le bas". Dans le cas contraire :

- Si D ne vaut ni 4, ni 3, ni 1, c'est que D vaut 2 et le programme doit sauter au bloc "Déplacement vers la droite".

On peut remarquer que cette méthode permet en même temps une protection implicite puisqu'un des quatre sauts se fera obligatoirement, même si le paramètre D fourni est erroné.

Nous n'aurons pas à nous préoccuper de transformer D en binaire. Celui-ci est en effet le langage naturel de l'ordinateur, et il ne travaille qu'avec lui lorsque nous employons l'hexadécimal, c'est par simple commodité. La première chose que fait donc l'ordinateur, c'est de traduire l'hexadécimal en binaire.

Pour tester les bits d'une valeur, nous utiliserons l'instruction : "Tester le bit numéro tant de tel registre". En l'occurrence, nous chargerons D dans le registre B. Il faut enfin savoir que cette instruction met l'indicateur de 0 si le bit testé vaut 0, et met l'indicateur de "non 0" si le bit testé vaut 1.

Après ces explications, les lignes 2 à 9 se passent presque de commentaires :

Ligne 2

Chargement de B avec le paramètre D.

Ligne 3

Test du bit 2 de B.

Ligne 4

Saut de +120 (ligne 69 : déplacement gauche) si non nul.

Ligne 5

Test du bit 1 de B.

Ligne 6

Saut de +40 (ligne 27 : déplacement haut) si nul.

Ligne 7

Test du bit 0 de B.

Ligne 8

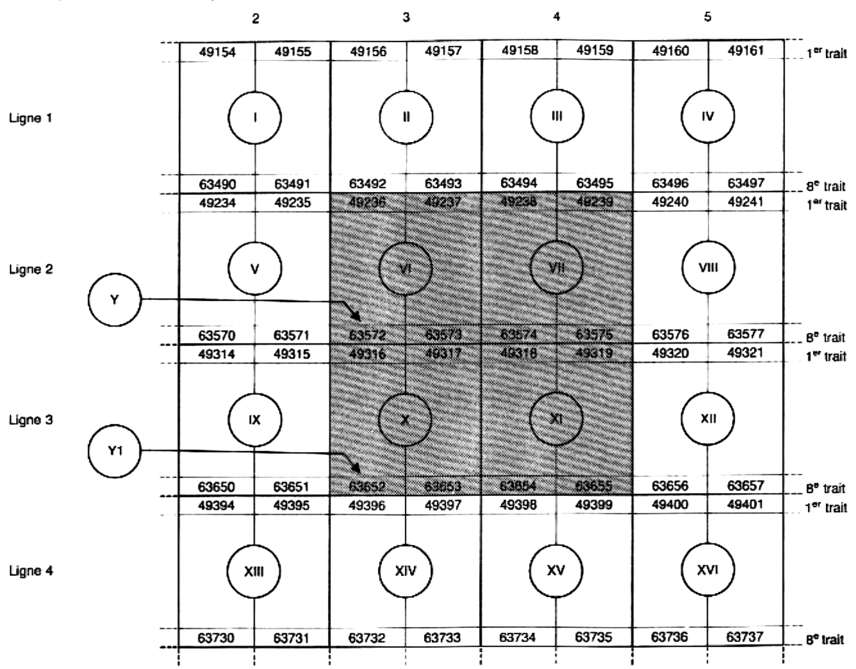
Saut de +2 (ligne 10 : déplacement bas) si non nul.

Ligne 9

Saut de +68 (ligne 44 : déplacement droit).

Nous allons maintenant étudier le bloc de lignes 10 à 26 qui effectue le déplacement vers le bas. Pour suivre les explications, référez-vous systématiquement à la figure suivante. Sur cette figure, la position initiale du mobile, choisie arbitrairement, est indiquée par la zone hachurée. Les lignes et colonnes indiquées (qui ne sont vraies que pour le mode 1), ainsi que les numéros d'octets correspondants, ne servent

que d'exemple et le raisonnement est valable quelle que soit la position initiale du mobile. D'autre part, et pour éviter la surcharge du dessin, seuls les premiers et huitième traits de chaque ligne ont été représentés. Enfin, l'ensemble est divisé en 16 blocs de 16 octets chacun, qui représentent en fait l'intersection d'une ligne et d'une colonne ou, si l'on préfère, une position de caractère en mode 1.



Notre mobile s'inscrivant initialement en VI, VII, IX et X, le déplacer vers le bas revient à effectuer les opérations suivantes :

1. Transférer X dans XIV
2. Transférer XI dans XV
3. Transférer VI dans X
4. Transférer VII dans XI
5. Effacer VI et VII

Le sous-bloc constitué des lignes 10 à 18 réalise les deux premières opérations.

Nous nous contenterons d'admettre pour l'instant que, quelle que soit la position initiale du mobile, les valeurs Y et Y1 qui représentent respectivement l'octet bas gauche (obg) du bloc haut gauche du mobile (VI pour le moment) et l'obg du bloc bas gauche du mobile nous sont connues et stockées dans leurs placards.

Ligne 10

Y1 est chargé dans HL.

Ligne 11

80 est chargé dans DE.

Ligne 12

Sauvegarde de Y1.

Ligne 13

Addition de HL et DE. Le résultat se retrouve dans HL.

Ligne 14

Échange de HL et DE. Ce dernier est donc maintenant chargé avec Y1+80. Rappelons (voir Annexe III) que pour passer d'un octet d'une ligne à l'octet de même situation mais une ligne plus bas, il suffit d'ajouter 80 au premier. Puisque DE est chargé avec Y1+80, il est donc pointé sur l'obg de XIV (63732 sur la figure).

Ligne 15

Y1 est remis dans HL.

Ligne 16

Ou "l'art d'être prévoyant" ; quand le mobile aura été déplacé d'une ligne vers le bas, à la fin du programme, Y et Y1 devront également être descendus d'une ligne pour suivre le mouvement. Sur notre exemple, Y vaudra 63652 et Y1 vaudra 63732. Or cette dernière valeur est actuellement dans DE. Au passage, nous en profitons donc pour charger Y1 dans le placard.

Ligne 17

Même chose, mais pour la future valeur de Y qui se trouve actuellement chargée dans HL. Remarquons que cette future valeur de Y est égale à la valeur actuelle de Y1.

Ligne 18

Appel du sous-programme interne d'adresse 43824 (ligne 95). Voyons ce sous-programme :

Lignes 95 et 96

Chargement de 8 dans A et de 4 dans BC. Ce sont là deux compteurs qui vont nous servir pour le transfert (pour transférer X et XI dans XIV et XV), il faut en effet transférer les huit traits de 4 octets chacun qui les composent.

Ligne 97

Transfert répétitif avec incrémentation (cette instruction a déjà été étudiée plusieurs fois).

A la fin de cette opération, HL se retrouve pointé sur l'obg de XII (63656 sur l'exemple), et DE sur l'obg de XVI (63736). En outre, le huitième trait de X et XI a été recopié (ou transferé) sur le huitième trait de XIV et XV. Nous pouvons maintenant effectuer la même opération avec le septième trait (qui n'est pas représenté sur l'exemple).

Lignes 98 et 99

Décrémentation de A, puis retour de sous-programme si nul. Avant de passer au septième trait, nous vérifions, grâce au compteur, combien de traits ont déjà été transférés. Si les huit l'ont été, le retour en ligne 19 est effectué, sinon le sous-programme continue en séquence.

Pour transférer le trait suivant, nous devons au préalable pointer HL sur l'octet situé immédiatement au-dessus de l'obg de X. Il n'est pas représenté, mais son adresse est 63652−2048=61604 (là encore, voir l'Annexe III). De même, DE doit être pointé sur l'octet immédiatement au-dessus de l'obg de XIV (=63732−2048=61684). On se rend compte que cela est possible en enlevant 2052 à la valeur actuelle de HL et DE (63656−2052=61604 et 63736−2052=61684).
C'est ce que vont faire les cinq lignes qui suivent :

Ligne 100

Chargement de 2052 dans BC.

Ligne 101

BC est soustrait de HL (le résultat est dans HL et BC reste inchangé).

Ligne 102

Échange de HL et DE.

Ligne 103

BC est soustrait de HL.

Ligne 104

Échange de HL et DE qui sont donc maintenant correctement pointés au début du septième trait de X et XIV.

Ligne 105

Saut relatif de −18 (ligne 96). Là, le compteur de transfert répétitif est à nouveau chargé avec 4, puis le transfert a lieu, A est décrémenté, et ainsi de suite. Après huit tours de boucle, au terme desquels les huit traits de X et XI auront été transférés, le retour de sous-programme sera effectué (en ligne 19).

A ce moment-là, HL et DE seront respectivement pointés sur l'octet haut gauche (ohg) de XII et XVI (49320 et 49400). Or nous voulons maintenant transférer VI et VII en X et XI. Pour cela, nous allons procéder exactement de la même manière que précédemment, et il nous faut donc pointer HL et DE respectivement sur l'obg de VI (63572) et sur l'obg de X (63652). Cela est possible en ajoutant 14252 à ces deux registres (49320+14252=63572 et 49400+14252=63652). C'est ce que font les lignes 19 à 23 : 14252 dans BC, HL+DE, échange de HL et DE, HL+DE, échange de HL et DE.

Ligne 24

Appel du même sous-programme interne que précédemment. Au retour de ce sous-programme, VI et VII auront été transférés dans X et XI, HL sera pointé sur 49240 et DE sur 49320. Il reste à effacer VI et VII.

Ligne 25

Le sous-programme interne d'adresse 43864 qui est appelé ici va se charger de cet effacement.

Ligne 117

Chargement de DE avec 2052. Nous verrons pourquoi plus loin.

Lignes 118 et 119

Chargement de 8 dans A et de 4 dans B. Ils serviront de compteurs : il y a huit traits de 4 octets chacun à effacer.

Ligne 120

Décrémentation de HL. Comme ce dernier était pointé sur 49240, il l'est maintenant sur 49239.

Ligne 121

La valeur 0 est chargée dans l'emplacement mémoire adressé par HL (ou si l'on veut, l'emplacement 49239 de la mémoire écran est mis à 0, donc effacé).

Ligne 122

Décrémentation de B et saut de −5 (ligne 120) si non nul. Les lignes 120 et 121 vont donc être exécutées quatre fois, HL pointant successivement sur 49239, 49238, 49237 et 49236 et les effaçant. Cela fait, B vaudra 0 et le sous-programme se poursuivra en séquence.

Lignes 123 et 124

Décrementation de A et retour si nul. Nous vérifions le nombre de traits effacés. Si les huit ne l'ont pas été, le sous-programme se poursuit.

Nous voulons maintenant effacer le trait suivant, c'est-à-dire pour l'instant le trait n° 7. Pour cela, et pour pouvoir utiliser le même processus que pour le huitième, nous devons pointer HL sur l'octet situé en dessous de 49240 (=49240+2048=51288). Or HL est actuellement chargé avec 49236. Il suffit donc de lui ajouter 2052, c'est-à-dire la valeur que nous avions mise dans DE à la ligne 117, et qui y est toujours.

Ligne 125

Addition de HL et DE (le résultat est dans HL et DE reste inchangé). HL est maintenant correctement pointé.

Ligne 126

Saut relatif de −12 (ligne 119). Ainsi la boucle va tourner tant que les huit traits de VI et VII n'auront pas été effacés.
Cette ligne est la dernière du sous-programme d'effaçage.
Le retour se fait en ligne 26.

Ligne 26

Le bloc "Déplacement vers le bas" étant terminé, retour au BASIC.

Le bloc "Déplacement vers le haut", qui va des lignes 27 à 43, effectue les opérations suivantes :

1. Transférer VI et VII dans II et III.
2. Transférer X et XI dans VI et VII.
3. Effacer X et XI.

Sans être tout à fait identique, il suit la même démarche que le bloc précédent, et vous ne devriez pas avoir de difficulté à l'étudier seul, pour peu que vous y alliez lentement et en prenant le temps de réfléchir.

Nous nous contenterons donc de donner la traduction en clair de chaque ligne :

Ligne 27 Y est chargé dans HL.

Ligne 28 80 est chargé dans DE.

Ligne 29 Empilement de Y.

Ligne 30 HL moins DE.

Ligne 31 Échange de HL et DE.

Ligne 32 Empilement de HL.

Ligne 33 Chargement de l'emplacement mémoire 43900 avec DE.

Ligne 34 Chargement de l'emplacement mémoire 43898 avec HL.

Ligne 35 Appel du sous-programme d'adresse 43824.

Ligne 36 Chargement de BC avec 14412.

Ligne 37 HL plus BC.

Ligne 38 Échange de HL et DE.

Ligne 39 HL plus BC.

Ligne 40 Échange de HL et DE.

Ligne 41 Appel du sous-programme d'adresse 43824.

Ligne 42 Appel du sous-programme d'adresse 43864.

Ligne 43 Retour de sous-programme.

Passons maintenant au bloc "Déplacement vers la droite" (lignes 44 à 68).

L'ordre des opérations qu'il convient de faire, indiqué ci-dessous, est un peu différent de celui des blocs précédents :

1. Transférer X et XI dans XI et XII.
2. Effacer X.
3. Transférer VI et VII dans VII et VIII.
4. Effacer VI.

Les lignes 44 à 56 réalisent les deux premières opérations :

Ligne 44

HL est pointé sur Y1.

Lignes 45 et 46

Deux incrémentations successives qui ont pour effet de pointer HL sur l'obg de XI (63654 sur notre exemple).

Ligne 47

Rangement de la future valeur de Y1 dans son placard.

Ligne 48

Pointage de HL sur l'obd (octet bas droit) de XI (63655 sur notre exemple).

Lignes 49 et 50

Une manière rapide de charger DE avec la même valeur que HL. Ils sont maintenant tous deux pointés sur l'obd de XI.

Lignes 51 et 52

DE est pointé sur l'obd de XII.

Ligne 53

Appel du sous-programme interne d'adresse 43844 (&AB44). Voyons ce sous-programme :

Lignes 106 et 107

Chargement de 8 dans A, puis de 4 dans BC. Vous avez compris qu'il s'agit là de l'initialisation des compteurs (pour transférer les blocs X et XI, il faut transférer les huit traits de 4 octets chacun qui les composent).

Ligne 108

Transfert répétitif avec décrémentation. Après l'exécution de cette ligne, le trait concerné a été décalé de deux positions vers la droite. Sur notre exemple, et après le premier tour de boucle concernant le huitième trait, HL se retrouve pointé sur 63651 et DE sur 63653.

Il faut maintenant transférer le trait suivant, situé juste au-dessus de celui-là, après avoir vérifié, grâce aux lignes 109 (décrémentation de A) et 110 (retour si nul), que les huit traits n'ont pas été transférés.

Vous avez sans doute déjà compris le processus : le pointage de HL et DE pour le transfert du trait suivant s'obtient en enlevant 2044 à HL et DE : 63653−2044=61609 (juste au-dessus de 63657), et 63651−2044=61607 (juste au-dessus de 63655).

Cette soustraction est effectuée par les lignes 111 à 115. Puis le saut relatif de la ligne 116 renvoie ensuite en début de boucle. Là, le compteur de transfert est réinitialisé, puis le trait est transféré, le compteur général décrémenté, etc.

Lorsque les huit traits sont transférés, le retour de sous-programme est effectué et l'on revient à la ligne 54. A ce moment précis, HL est pointé sur l'ohd (octet haut droit) de IX (49315 sur notre exemple), et DE sur l'ohd de X (49317 sur l'exemple).

Lignes 54 et 55

Nous aurons besoin ultérieurement des valeurs contenues dans HL et DE, et nous les sauvegardons par empilement.

Ligne 56

Appel du sous-programme interne d'adresse 43881 (&AB69), ligne 127. C'est lui qui va se charger de l'effaçage du bloc X.

Ligne 127

Chargement de DE avec 2046. Nous verrons pourquoi plus loin.

Lignes 128 et 129

Chargement de 8 dans A et de 4 dans B : les compteurs sont initialisés (il y aura huit traits de 2 octets chacun à effacer).

Ligne 130

Incrémentation de HL. Rappelons que ce dernier était toujours pointé sur l'ohd de IX. Il l'est maintenant sur l'ohg de X (49236 sur l'exemple).

Ligne 131

Chargement de 0 dans l'emplacement mémoire adressé par HL ou, si l'on veut, effacement de cet emplacement.

Ligne 132

Décrémentation de B et saut relatif de −5 (en ligne 129) si non nul. Les lignes 129 et 130 vont donc être exécutées deux fois, HL pointant d'abord sur l'ohg puis l'ohd de X (49316 et 49317 sur notre exemple). Lorsque cela est terminé, donc lorsque le premier trait est effacé, le sous-programme continue en séquence.

Lignes 133 et 134

Décrémentation de A et retour si nul. Le nombre de traits déjà effacés est vérifié. Si les huit l'ont été, le retour est effectué (en ligne 57), sinon le sous-programme continue en séquence.

Ligne 135

Pour effacer le trait suivant, HL doit être pointé comme précédemment, mais un trait plus bas. Il suffit pour cela d'ajouter 2046 à sa valeur actuelle (49317 + 2046=51363). Or DE avait justement été chargé avec 2046 en ligne 127, et il l'est toujours. Il suffit donc d'ajouter DE et HL, comme le fait cette ligne (rappelons que le résultat se retrouve dans HL et que DE reste inchangé).

Ligne 136

Saut relatif de −12 (en ligne 129). Là, le compteur B est réinitialisé, HL est incrémenté, l'emplacement qu'il adresse effacé, etc.

Lorsque ce sous-programme a terminé sa tâche et que X est effacé, le retour se fait en ligne 57. Il nous faut maintenant transférer VI et VII dans VII et VIII, puis effacer VI.

Lignes 57 et 58

Dépilement des registres DE puis HL qui se retrouvent donc respectivement pointés sur l'ohd de X et l'ohd de IX. Mais ce que nous voulons obtenir, c'est leur pointage sur l'obd de VIII et l'obd de VII. On atteint ce résultat en leur ajoutant 14260 (49317+14260=63577 et 49315+14260=63575). Cette opération est réalisée par les lignes 59 à 63. Leur principe vous est maintenant familier, et nous n'y reviendrons pas.

La ligne 64 appelle ensuite le sous-programme d'adresse 43844, déjà étudié, et qui réalise le transfert. La ligne 65 appelle le sous-programme d'adresse 43881 qui efface le bloc VI.

Au retour de ce dernier sous-programme, HL se retrouve pointé sur l'obd de VI.

Ligne 66

Incrémentation de HL qui contient donc maintenant l'adresse de l'obg de VII, c'est-à-dire la valeur de Y après le déplacement du mobile.

Ligne 67

Cette valeur est rangée dans son placard.

Ligne 68

Le déplacement vers la droite étant terminé, le retour au BASIC est effectué.

Reste le bloc "Déplacement vers la gauche", qui va des lignes 69 à 94. Il s'effectue d'une manière comparable au déplacement vers la droite, et cela constituera un excellent exercice pour vous de l'étudier seul.

Avant d'en avoir tout à fait terminé, il nous faut encore élucider l'énigme des adresses Y et Y1.

Nous avons considéré jusqu'à présent que ces valeurs étaient connues et, qui plus est, rangées dans leurs placards au moment de l'appel du programme.

Il est bien évident qu'elles n'y viennent pas toutes seules, et qu'il faudra les charger avant de pouvoir utiliser le programme de déplacement.

Regardez à nouveau la figure : le mobile est situé au départ sur quatre emplacements dont les coordonnées sont (3,2), (4,2), (3,3) et (4,3). Or, l'adresse de l'obg d'un emplacement donné peut fort bien être calculée en fonction des coordonnées de cet emplacement :

$$\text{obg d'un emplacement} = 63406 + (80 \times \text{n° de ligne}) + (2 \times \text{n° de colonne})$$

Soit, dans notre cas :

Y = 63406 + (80 * 2) + (3 * 2) = 63572

et

Y1 = 63406 + (80 * 3) + (3 * 2) = 63652

Une fois ces adresses calculées, rien n'est plus facile que de les ranger dans leurs placards. Convertissons-les d'abord en hexadécimal :

HEX$(63572) = F854
HEX$(63652) = F8A4

Rappelons que, lorsque l'on range un nombre en mémoire, l'octet faible est toujours rangé en premier. Voici donc la ligne BASIC qui range Y et Y1 :

```
POKE 43898,&A4 : POKE 43899,&F8 : POKE 43900,&54 :
        POKE 43901,&F8
```

Il n'est nécessaire d'exécuter cette ligne qu'une seule fois avant de pouvoir faire appel au programme de déplacement, puisque Y et Y1 sont sauvegardés au fur et à mesure que le mobile se déplace.

Bien entendu, la formule proposée plus haut est valable, en mode 1, quelque soit l'endroit initial où vous souhaitez positionner le mobile.

Ce programme n'a maintenant plus de secrets pour vous et, fidèles à la coutume, nous vous proposons ci-dessous deux exemples de son utilisation :

```
100 POKE 43898,&A4 : POKE 43899,&F8 :
    POKE 43900,&54 : POKE 43901,&F8
110 MODE 1 : BORDER 3
120 FOR I=3 TO 4 : FOR J=2 TO 3 : LOCATE I,J : PRINT
    CHR$(206) : NEXT J,I
130 A=3 : B=2
140 IF INKEY(0)=0 THEN IF B-1=0 THEN 140 ELSE CALL
    43650,1 : B=B-1
150 IF INKEY(2)=0 THEN IF B+1=25 THEN 140 ELSE CALL
    43650,3 : B=B+1
160 IF INKEY(8)=0 THEN IF A-1=0 THEN 140 ELSE CALL
    43650,4 : A=A-1
170 IF INKEY(1)=0 THEN IF A+1=40 THEN 140 ELSE CALL
    43650,2 : A=A+1
180 GOTO 140
```

La ligne 100 range les adresses Y et Y1 ; la ligne 120 dessine le mobile ; les lignes 130 à 180 permettent de déplacer le mobile en utilisant les flèches du curseur (notons qu'une protection a été installée pour empêcher le mobile de sortir de l'écran, ce qui pourrait provoquer des résultats fâcheux. Cette protection pourrait fort bien être installée directement dans le programme en langage machine. Si le cœur vous en dit...).

Enfin, pour les fanatiques d'astéroïdes, vaisseaux spatiaux et autres fusées, voici une modeste tentative graphique (la ligne 120 est remplacée et deux autres sont insérées) :

```
120 FOR I=49237 TO 63573 STEP 2048 : READ A : POKE
    I,A : READ A : POKE I+1,A : NEXT I
122 FOR I=49316 TO 63652 STEP 2048 : FOR J=0 TO 3 :
    READ A : POKE I+J,A : NEXT J,I
124 DATA 17, 136, 17, 136, 17, 136, 51, 204, 48, 192,
         119, 238, 48, 192, 48, 192, 0, 7, 14, 0, 0, 55,
         206, 0, 1, 63, 207, 8, 3, 63, 207, 12, 3, 95,
         110, 12, 2, 102, 102, 4, 2, 96, 96, 4, 2, 96,
         96, 4
```

Ce dernier exemple a surtout pour objet de vous montrer que le programme n'est pas tributaire de l'instruction LOCATE et que, en ce sens, le mode est indifférent.

8. EXEMPLE 8 : TRI DE DONNÉES ALPHANUMÉRIQUES

Notre approche du langage machine serait incomplète si elle ne comprenait pas une petite incursion dans le royaume des données alphanumériques. (L'étude préalable de l'Annexe V, section 2, est impérative.)

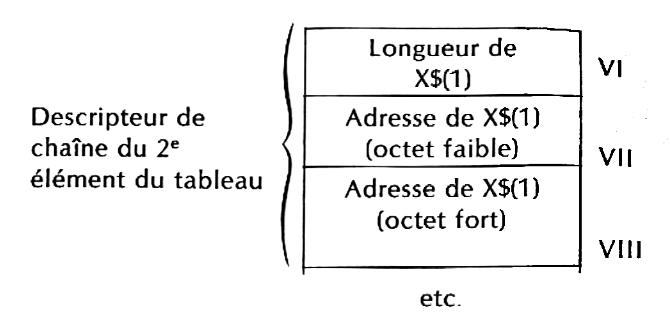
Le programme que nous vous proposons maintenant permet de ranger un tableau de données alphanumériques dans l'ordre alphabétique. Il débute à l'adresse 43830, finit en 43896 (chiffre de vérification : 7168), n'est pas directement relogeable, et son format d'appel est le suivant :

CALL 43830, @ X$(0)

X$ est le nom du tableau (qui peut être quelconque), et l'ensemble @ X$(0) pointe donc sur l'adresse du descripteur de chaîne de la première donnée du tableau (son rang est 0).

Il est indispensable que nous nous arrêtions un moment sur cette notion de tableau et, pour être plus précis, sur l'organisation en mémoire des descripteurs de chaîne des éléments qui le composent.
Pour n'importe quel tableau X$ (à une seule dimension), ces descripteurs de chaîne sont stockés de la manière suivante :

Nombre d'éléments du tableau X$	Octet faible	I
	Octet fort	II
Descripteur de chaîne du 1er élément du tableau	Longueur de X$(0)	III
	Adresse de X$(0) (octet faible)	IV
	Adresse de X$(0) (octet fort)	
		V

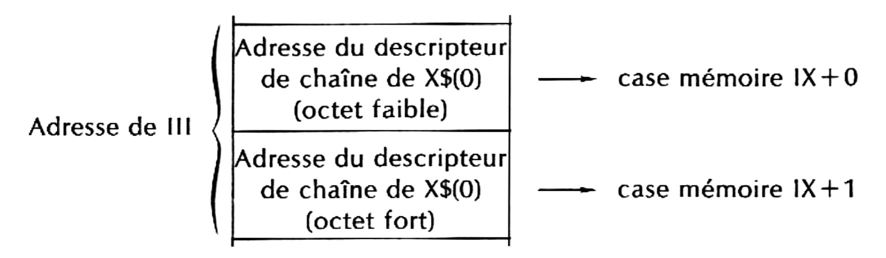


Les chiffres romains indiqués sur la droite ne nous serviront qu'à nommer les emplacements mémoire correspondants lors des explications.

Rappelons que l'appel du programme se faisant sous le format :

CALL 43830, @ X$(0)

Le registre IX sera pointé comme suit :



Il sera nécessaire, tout au long des explications, de vous reporter à ces deux figures.

Voyons maintenant le principe de tri adopté. Il s'agit de la technique dite du *tri par bulles*, dont le processus est très simple : notre tableau étant constitué d'une série de chaînes de caractères, on pointe sur l'une d'elles, et l'on compare le code ASCII (voir à ce sujet la page A3.1 du guide de l'utilisateur) de sa première lettre au code ASCII de la première lettre de la chaîne de caractères suivante.

Le code ASCII le plus petit indique que la lettre correspondante est située avant dans l'alphabet, et les deux chaînes de caractères sont ou ne sont pas inversées, selon le cas.

Si une inversion est nécessaire, elle est effectuée, puis le pointeur est repositionné au début du tableau et le processus est repris.

Si le pointeur arrive en fin de tableau sans qu'aucune inversion ait

été nécessaire, c'est donc qu'il n'y a plus d'élément à inverser et que le tableau est rangé.

Le nom de *tri par bulles* s'explique par une analogie : à chaque passe du pointeur, les éléments les plus "légers" remontent progressivement à la surface.

Pour ce qui nous concerne, précisons que, lorsqu'il sera nécessaire d'inverser deux données du tableau, nous nous contenterons en fait d'inverser leur descripteurs de chaîne respectifs.

Si tout cela est bien clair, nous pouvons passer à l'étude du listing.

LIGNE	ADRESSE DU 1er CODE DE CETTE LIGNE DEC.	HEXA.	CODE-MACHINE	MNEMONIQUE
1	43830		DD,66,1	LD H,(IX+1)
2	43833		DD,6E,0	LD L,(IX+0)
3	43836		2B	DEC HL
4	43837		46	LD B,(HL)
5	43838		2B	DEC HL
6	43839		4E	LD C,(HL)
7	43840		B	DEC BC
8	43841		C5	PUSH BC
9	43842		23	INC HL
10	43843		23	INC HL
11	43844		23	INC HL
12	43845		5E	LD E,(HL)
13	43846		23	INC HL
14	43847		56	LD D,(HL)
15	43848		23	INC HL
16	43849		23	INC HL
17	43850		4E	LD C,(HL)
18	43851		23	INC HL
19	43852		46	LD B,(HL)
20	43853		A	LD A,(BC)
21	43854		EB	EX HL,DE
22	43855		BE	CP (HL)
23	43856		FA,5D,AB	JP M,43869
24	43859		E1	POP HL
25	43860		2B	DEC HL
26	43861		7C	LD A,H
27	43862		B5	OR L
28	43863		C8	RET Z
29	43864		E5	PUSH HL
30	43865		1B	DEC DE
31	43866		EB	EX HL,DE
32	43867		18,E8	JR -24
33	43869	AB5D	E1	POP HL
34	43870		D5	PUSH DE
35	43871		E1	POP HL
36	43872		1A	LD A,(DE)
37	43873		F5	PUSH AF
38	43874		1B	DEC DE
39	43875		1A	LD A,(DE)
40	43876		F5	PUSH AF
41	43877		1B	DEC DE
42	43878		1A	LD A,(DE)

```
43    43879    EB       EX HL,DE
44    43880    2B       DEC HL
45    43881    1,3,0    LD BC,3
46    43884    ED,B8    LDDR
47    43886    EB       EX HL,DE
48    43887    13       INC DE
49    43888    12       LD (DE),A
50    43889    F1       POP AF
51    43890    13       INC DE
52    43891    12       LD (DE),A
53    43892    F1       POP AF
54    43893    13       INC DE
55    43894    12       LD (DE),A
56    43895    18,BD    JR -67
```

Le premier bloc remarquable va des lignes 1 à 8, et initialise le compteur. Nous aurons en effet besoin d'être avertis si tout le tableau est parcouru sans qu'il y ait d'inversion, signe que le programme est terminé. En réalité, il suffira d'atteindre l'avant-dernier élément du tableau, puisqu'il aura alors été comparé au suivant, c'est-à-dire au dernier. La valeur qui va nous intéresser est donc :

(nombre d'éléments du tableau) – 1

Lignes 1 et 2

Chargement de HL avec (IX+0) et (IX+1). Le registre HL est ainsi pointé sur III (voir la figure).

Ligne 3

Décrémentation de HL qui est maintenant pointé sur II (gardez bien à l'esprit les pointages successifs de HL).

Ligne 4

Chargement du contenu de l'emplacement mémoire sur lequel est pointé HL dans le registre B. Ce dernier contient donc maintenant l'octet fort du nombre d'éléments du tableau.

Lignes 5 et 6

Même principe pour charger l'octet faible dans C. Après cette ligne, BC contient le nombre d'éléments du tableau.

Lignes 7 et 8

Décrémentation de BC (puisque nous voulons obtenir la valeur (nombre d'éléments –1)), puis sauvegarde par empilement.

Le deuxième bloc va des lignes 9 à 23 et effectue les tâches suivantes : initialisation des pointeurs, démarrage des passes, comparaison des éléments et branchement en fonction du résultat :

Lignes 9, 10 et 11

Trois incrémentations successives de HL qui se retrouve pointé sur IV.

Lignes 12, 13 et 14

Suivant un principe analogue à celui des lignes 4, 5 et 6, l'adresse du premier élément du tableau est chargée dans DE.

Lignes 15 à 19

Toujours selon le même principe, l'adresse de l'élément suivant du tableau est chargé dans BC.

Récapitulons la situation :

- DE est pointé sur l'emplacement mémoire contenant le code ASCII de la première lettre du premier élément du tableau.
- BC est pointé sur l'emplacement mémoire contenant le code ASCII de la première lettre du deuxième élément du tableau.
- HL est pointé sur VIII.
- Le nombre d'éléments du tableau − 1 est sur la pile.

Ligne 20

Le contenu de l'emplacement mémoire adressé par BC est chargé dans A, qui contient donc maintenant le code ASCII de la première lettre du deuxième élément du tableau.

Ligne 21

Échange de DE et HL. Ce dernier se retrouve pointé sur l'emplacement mémoire contenant le code ASCII de la première lettre du premier élément du tableau, et DE se retrouve pointé sur VIII.

Ligne 22

Comparaison entre A et (HL), donc entre les deux codes ASCII.

Rappelons brièvement le principe de cette comparaison (qui a déjà été étudiée dans d'autres programmes).

(HL) est soustrait de A, sans que le résultat soit conservé, c'est-à-dire sans que le contenu des deux registres soit modifié. Les indicateurs S et Z sont mis en fonction du résultat.

En l'occurrence, si le code ASCII du premier élément est plus petit que celui du deuxième élément (donc si la situation alphabétique des deux éléments l'un par rapport à l'autre est correcte), l'indicateur S indiquera un résultat positif.

Dans le cas contraire, S indiquera un résultat négatif et nous devrons inverser les descripteurs de chaîne des 2 éléments.

Ligne 23

Saut à l'adresse 43869 (ligne 33) si négatif. A cette adresse commence en effet le bloc chargé de l'inversion. Voyons d'abord le cas où cette inversion n'est pas nécessaire et où le saut est ignoré :

Ligne 24

L'état actuel du compteur est récupéré et chargé dans HL.

Lignes 25, 26 et 27

Après une décrémentation de HL, on vérifie si son contenu est ou n'est pas égal à 0 (selon un principe déjà étudié dans le programme de pause, lignes 6, 7 et 8).

Ligne 28

Retour de sous-programme si nul : si HL = 0, donc si le tableau a été parcouru jusqu'à l'avant-dernier élément sans qu'il y ait eu besoin d'inversion, le tableau est rangé correctement, et le programme est terminé. Sinon, cette instruction est ignorée.

Ligne 29

Sauvegarde de la nouvelle valeur du compteur.

Ligne 30

Décrémentation de DE qui, depuis la ligne 21, était pointé sur VIII et l'est maintenant sur VII.

Ligne 31

Échange de HL et DE : HL est pointé sur VII.

Ligne 32

Saut relatif de – 24 en ligne 12. La boucle va donc reprendre, mais le pointage dans le tableau sera décalé d'un élément (au premier tour, la comparaison se fera entre les éléments 1 et 2, au deuxième tour entre les éléments 2 et 3, au quatrième tour entre les éléments 3 et 4, etc.).

Si cette boucle est exécutée (nombre d'éléments – 1) fois sans inversion, le programme retournera au BASIC en ligne 28. Mais attention : à chaque inversion, *le processus doit reprendre depuis le début,* et le compteur devra être en particulier réinitialisé.

Nous allons maintenant passer au bloc de lignes 33 à 56, qui se charge des éventuelles inversions des descripteurs de chaîne.

Pour servir d'exemple, nous prendrons le cas où les descripteurs de chaîne des deux premiers éléments du tableau doivent être inversés. Pour ce faire, il faut exécuter les tâches suivantes :

1. Sauvegarder les contenus de VI, VII et VIII.
2. Transférer les contenus de III, IV et V respectivement dans VI, VII et VIII.
3. Mettre les valeurs sauvegardées (contenus de VI, VII et VIII), respectivement dans III, IV et V.

Rappelons qu'au moment où le branchement vers ce bloc se fait, en ligne 23, DE était pointé sur VIII.

Ligne 33

La valeur actuelle du compteur ne nous intéresse plus, puisque celui-ci devra être réinitialisé. C'est pourquoi nous l'enlevons de la pile.

Lignes 34 et 35

HL est, comme DE, pointé sur VIII.

Lignes 36 à 42

Elles réalisent la sauvegarde des contenus de VI, VII et VIII. Ces lignes ne nécessitent aucune explication particulière : vous connaissez toutes les instructions utilisées, et il suffit de suivre attentivement, si possible en les notant, l'état de la pile et l'évolution du contenu des registres.

Une remarque cependant : vous pourrez constater que sont utilisées les deux instructions suivantes : "PUSH AF" et "POP AF". Or, vous n'avez jamais encore entendu parler de ce registre double AF ! En réalité, seul le contenu de A nous intéresse, et pas du tout celui de F. Mais comme il est impossible d'empiler un registre simple, nous sommes obligés d'empiler le registre double AF, en nous disant que A l'est par la même occasion.
Quand vous rencontrerez ces deux instructions, vous pourrez faire comme s'il y avait "PUSH A" et "POP A" à la place.

Lignes 43 à 46

Comme précédemment, ce bloc, qui réalise le transfert des contenus de V, IV et III dans VIII et V, s'explique de lui-même. L'instruction de transfert répétitif avec décrémentation, déjà plusieurs fois étudiée, est en particulier utilisée.

Signalons également qu'après l'exécution de ces lignes, DE est pointé sur V et HL sur II.

Lignes 47 à 55

Là encore, rien de particuier. Ce bloc réalise évidemment le transfert des contenus initiaux de VIII, VII et VI dans V, IV et III. L'inversion des deux descripteurs de chaîne est terminée.

Ligne 56

Puisqu'une inversion a eu lieu, tout le processus doit être repris : saut relatif de – 67, en ligne 1.

Voici un exemple d'utilisation de ce programme :

```
100 DIM TABLEAU$ (24)
110 CLS : FOR I=0 TO 24 : A=INT(RND * 26)+65 :
    TABLEAU$(I)=CHR$(A) : NEXT
120 FOR I=0 TO 24 : PRINT TABLEAU$(I) : NEXT
130 CALL 43830, @TABLEAU$(0)
140 FOR I=0 TO 24 : LOCATE 20,I+1 : PRINT TABLEAU$(I) :
    NEXT
150 GOTO 150
```

La ligne 100 crée un tableau de 25 éléments.
La ligne 110 remplit aléatoirement ce tableau avec les lettres de l'alphabet (majuscules).

La ligne 120 affiche sur la gauche de l'écran le contenu initial du tableau.
La ligne 130 effectue le rangement.
La ligne 140 affiche, à droite de l'écran, le contenu du tableau après ce rangement.
La ligne 150 empêche le scrolling d'écran (il faut arrêter le programme en tapant 2 fois la touche "ESC").

Remarque

Le tableau n'est ici constitué que de lettres isolées, mais il pourrait tout aussi bien être constitué de mots.

A titre de comparaison, le programme en langage machine range le tableau en plus ou moins 1/10e de seconde, alors qu'il faudrait jusqu'à 200 fois plus de temps à un programme BASIC équivalent !

IV
ANNEXES

1. LES SAUTS RELATIFS

Il est possible, dans un programme en langage machine, d'exécuter ce que l'on appelle des sauts relatifs, qui peuvent se faire vers l'avant ou vers l'arrière.

Prenons l'exemple du petit programme suivant :

```
┌>21   - 6
│ 1    - 5
│ 0    - 4
│ 2B   - 3
│ 18   - 2
└─FA   - 1
```

Les trois premiers octets chargent HL avec 1. Le quatrième décrémente HL. Le cinquième indique un saut, et le sixième le sens et la longueur de ce saut.

Ici le saut est de − 6. Le programme va boucler sans arrêt, HL passant ainsi alternativement de 0 à 1 et de 1 à 0. Vous remarquerez que l'octet indiquant la longueur et le sens du saut (FA), est lui-même décompté.

Sa valeur en hexadécimal est déterminé par la formule suivante (uniquement valable pour les sauts en arrière) :

HEX$ (256 − longueur du saut souhaité)

Ici : HEX$ (256−6)= FA

Un saut de −31 donnerait par exemple : HEX$(256−31) = E1

La longeur maximale d'un saut en arrière est de −128.

Voyons maintenant le cas d'un saut en avant :

```
  21
  1
  0
  18
┌─3
│ 11
│ 7    + 1
│ 0    + 2
└>2B   + 3
```

Le quatrième octet indique un saut relatif et le cinquième détermine son sens et sa longueur. Cette fois, non seulement cet octet ne doit pas être compté, *mais le suivant non plus* (ici 11).

Pour un saut en avant, l'octet déterminant sa longueur se calcule comme suit :

HEX$ (longueur du saut)

Ici : HEX$(3) = 3

Un saut de 50 donnerait par exemple : HEX$(50) = 32

La longueur maximale d'un saut en avant est de 127.

Pour vous faciliter la tâche lorsque vous concevrez vos propres programmes, nous vous proposons ci-après deux tables : l'une pour les sauts en avant et l'autre pour les sauts en arrière.

Table des sauts en avant :

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	129	121	122	123	124	125	126	127

Table des sauts en arrière :

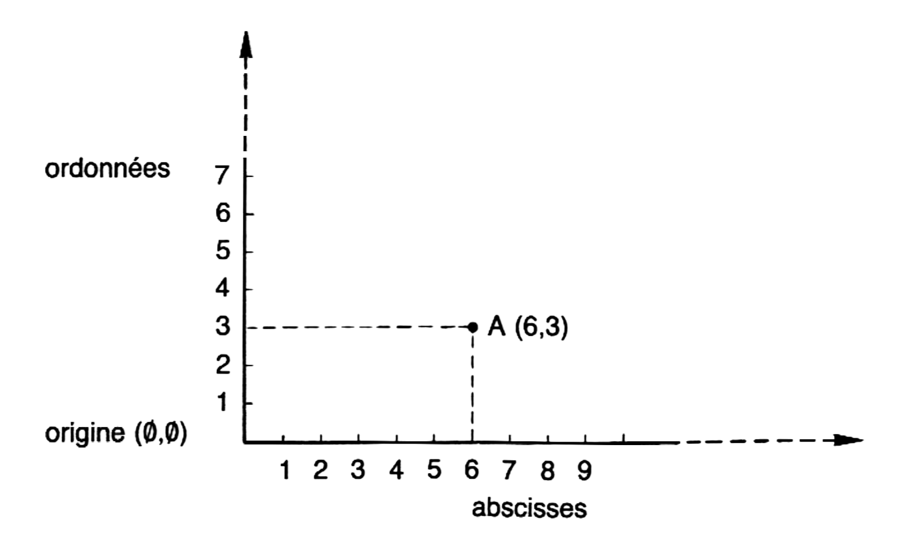
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

L'octet correspondant à une valeur de saut est constitué par la lettre ou le chiffre en regard de la valeur choisie dans le tableau, sur la colonne de gauche, et par la lettre ou le chiffre en regard sur la ligne du haut.

2. LES COORDONNÉES GRAPHIQUES RELATIVES ET ABSOLUES

LES COORDONNÉES ABSOLUES

L'écran du CPC peut être considéré comme un repère orthonormé, dont l'origine se situe au coin inférieur gauche, l'axe des X (abscisses) allant de 0 à 639, et l'axe des Y (ordonnées) de 0 à 399 :



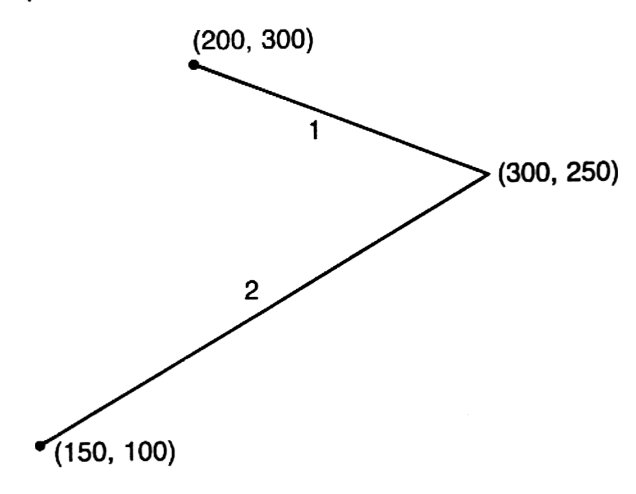
Les coordonnées dites absolues sont *toujours exprimées en fonction de l'origine*. Exemple : le point A, de coordonnées absolues (6,3).
Il faut savoir en outre que, lorsque vous allumez la machine, le curseur graphique (qui peut être considéré comme la pointe du crayon permettant de dessiner sur l'écran) est positionné en (0,0). Mais dès qu'un point quelconque est dessiné grâce à l'instruction BASIC PLOT, le curseur graphique se retrouve positionné sur ce point. La règle générale est que le curseur se trouve toujours sur le dernier point dessiné, ou à l'extrémité de la dernière ligne dessiné.

Les lignes peuvent être dessinées avec l'instruction BASIC DRAW : la ligne est tracée à partir de la position du curseur graphique à ce moment là, jusqu'au point dont les coordonnées absolues suivent l'instruction DRAW (il s'agit bien de coordonnées absolues, et l'extrémité de la ligne sera indépendante de la position du curseur.

Essayez par exemple de lancer le programme suivant, qui trace deux lignes :

```
10 MODE 1
20 PLOT 200,300
30 DRAW 300,250
40 PLOT 150,100
50 DRAW 300,250
```

Voici ce que vous obtenez :



- La ligne 20 dessine un point de coordonnées absolues (200,300). Par la même occasion, le curseur y est positionné.
- La ligne 30 trace une droite à partir du curseur graphique jusqu'à la position de coordonnées absolues (300,250). Il s'agit de la droite 1. Le curseur se trouve maintenant en (300,250).
- La ligne 40 dessine un point de coordonnées (150,250) et y fixe le curseur graphique.
- La ligne 50, enfin, trace une droite à partir du curseur graphique jusqu'à une position absolue (300,250).

Pour nos programmes en langage machine, nous utiliserons les deux routines suivantes :

&BBEA qui est l'équivalent de PLOT en BASIC (dessin d'un point)

&BBF6 qui est l'équivalent de DRAW en BASIC (tracé d'une ligne allant du curseur graphique jusqu'à la position absolue spécifiée).

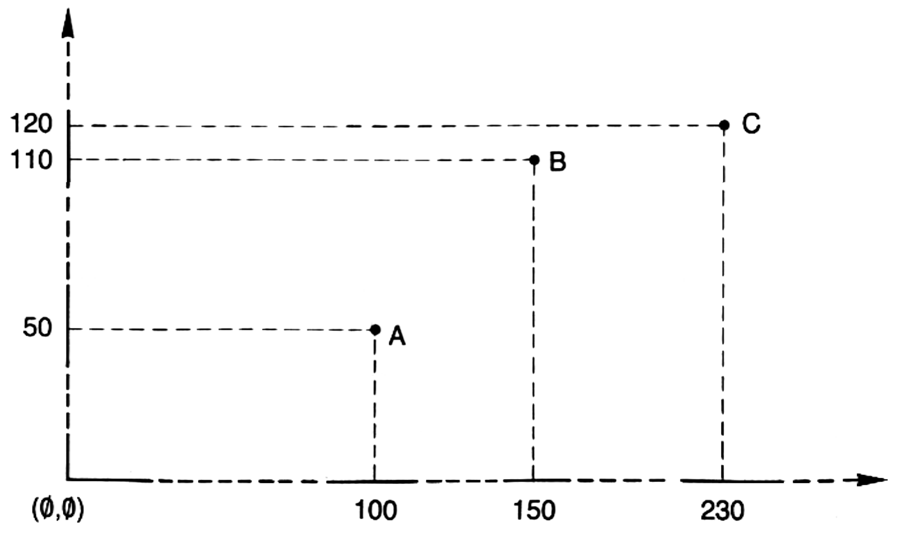
LES COORDONNÉES RELATIVES

Les coordonnées dites relatives sont *toujours exprimées en fonction de la position du curseur graphique à ce moment-là* (c'est-à-dire comme si la position de ce dernier représentait l'origine des axes).

Un point de coordonnées relatives peut être dessiné en BASIC grâce à l'instruction PLOTR.

Exemple

```
10 MODE 1
20 PLOT 100, 50 (dessin du point A, en coordonnées absolues)
30 PLOTR 50, 60 (dessin du point B, en coordonnées relatives)
40 PLOTR 80,10 (dessin du point C, en coordonnées relatives)
```



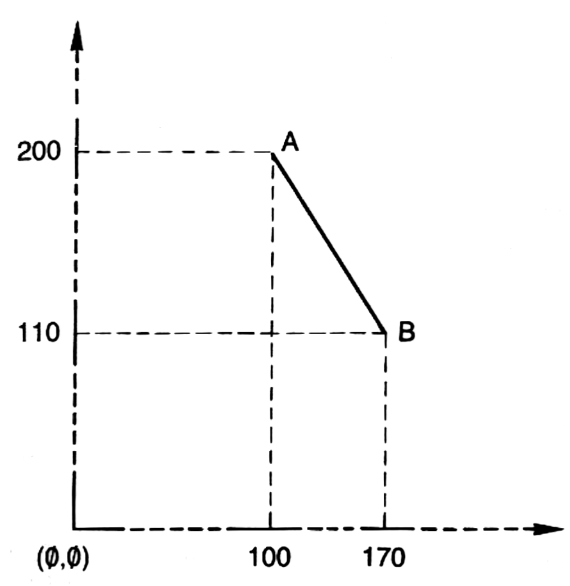
On peut effectivement constater que les coordonnées absolues de B sont (150,110), mais ses coordonnées relatives par rapport à A (où se

trouve le curseur quand B est dessiné) sont bien (50,60). D'une manière moins orthodoxe, on pourrait dire que B est plus à droite que A de 50, et plus haut de 60.
De la même manière, les coordonnées absolues de C sont (230,120), et ses coordonnées relatives par rapport à B sont (80,10).

Les "lignes relatives" peuvent être tracées grâce à l'instruction BASIC DRAWR : tracé d'une ligne à *partir de la position du curseur graphique* à ce moment là, jusqu'à une position dont les coordonnées *relatives à ce curseur* suivent l'instruction DRAW.

Exemple

10 MODE 1 : PLOT 100,200: DRAW 70,-90



L'extrémité B de la droite a pour coordonnées absolues (170,110), et pour coordonnées relatives par rapport à A (70,−90). Elle est en effet de 70 plus à droite que A, et de 90 plus basse que B.

En langage machine, nous utiliserons les deux routines suivantes :

&BBED qui est l'équivalent de PLOTR.

&BBF9 qui est l'équivalent de DRAWR.

Notons enfin qu'il est possible de positionner le curseur graphique à une position absolue donnée, sans rien dessiner, grâce à l'instruction BASIC MOVE, dont la routine équivalente, que nous utiliserons en langage machine, est &BBC0.

3. LA MÉMOIRE ÉCRAN

Pour simplifier cette étude, nous nous limiterons dans un premier temps au mode 1.

La mémoire écran est une zone de 16K (16384 octets) qui, lorsque vous allumez votre machine, va de l'adresse 49152 à l'adresse 65535. Le schéma de la page suivante montre son organisation.

Comme vous pouvez le constater, chaque colonne a 2 octets de large et chaque ligne 8 octets de haut, ce qui implique qu'en mode 1 un caractère tiendra sur 8 fois 2 = 16 octets (pour information, 1 octet comprend 8 bits ou, si l'on parle de l'écran, 8 pixels, ou encore 8 points de large).

L'écran du CPC est un peu particulier. Vous remarquerez en effet que la première rangée d'octets (ou si l'on veut le premier "trait" de la première ligne), commence à l'adresse 49152 et finit à l'adresse 49231. On pourrait logiquement s'attendre à ce que le deuxième trait commence à l'adresse 49232, ce qui n'est pas le cas (il commence en 51200). L'octet 49232 se trouve au début du premier trait de la deuxième ligne.

Les octets de la mémoire écran sont donc numérotés de la maniée suivante : d'abord les premiers traits des 25 lignes de l'écran, puis les deuxièmes traits, puis les troisièmes, et ainsi de suite.

On peut facilement vérifier cela en essayant ce petit programme :

```
10 MODE 1
20 FOR I=49152 TO 51151 : POKE I,255 : NEXT
30 FOR I=51200 TO 53199 : POKE I,150 : NEXT
```

La ligne 20 trace les premiers traits de toutes les lignes en rouge et la ligne 30 les deuxièmes traits de toutes les lignes en bleu clair.

Une curiosité subsiste néanmoins, qui ne vous a certainement pas échappé : le trait n° 1, par exemple, se termine en 51151 (25e ligne). Or le deuxième trait commence en 51200 (1re ligne). Manquent donc à l'appel 48 octets, et il en est de même à la fin de chaque trait, c'est-à-dire 8 fois.

En fait, ces 48 fois 8 = 384 octets ne sont pas perdus, puisque la machine s'en sert pour gérer par exemple les *scrollings* d'écran (=déplacements, dans l'une des quatre directions, de tout l'écran).

Cette disposition de la mémoire écran, pour particulière qu'elle soit, n'est en réalité guère plus difficile à travailler qu'une autre, dans la mesure ou l'on est averti.

En particulier, et parce que nous nous en servirons dans nos programmes, sachez que :

- Pour passer d'un trait à un autre *d'une même ligne*, on ajoute ou on enlève 2048 ou un de ses multiples. Par exemple, pour passer du deuxième au troisième trait de la ligne 1, il suffit de faire 51200+2048=53248. Cela est valable quelle que soit la position d'octet initiale.

- Pour passer sur le *même trait d'une autre ligne*, on enlève ou on ajoute 80 ou un des ses multiples. Par exemple, pour passer du premier trait de la première ligne au premier trait de la 24ᵉ ligne, il suffit de faire 49152+(23×80)=50992.

- Le numéro d'un octet précis peut toujours être déterminé à partir du numéro de colonne et du numéro de ligne correspondants. Par exemple, pour calculer l'octet bas/gauche de l'emplacement situé à l'intersection de la deuxième colonne et de la 24ᵉ ligne, il suffit de calculer 63406+(80×24)+(2×2). La formule générale est dans ce cas précis 63406 + 80Y + 2X).

Pour conclure, parlons un peu des différents modes possibles et de leurs différences (nous laisserons de côté le problème des couleurs, qui est plus complexe et nous éloignerait du sujet de ce livre).

Vous n'ignorez pas qu'il est possible, sur l'Amstrad, de travailler sur 20 colonnes en mode 0, sur 40 colonnes en mode 1, et sur 80 colonnes en mode 2 (le nombre de lignes reste inchangé : 25). Mais en terme d'octets, la configuration de la mémoire écran est la même pour les trois modes. L'écran ayant une largeur de 80 octets, le calcul est vite fait :

- En mode 0, chaque caractère est contenu dans une surface de 4 octets de large (sur la figure précédente, un caractère écrit en 1,1 s'inscrirait dans le rectangle déterminé par les octets 49152, 49155, 63488 et 63491).

- En mode 1, chaque caractère s'inscrit dans une surface de 2 octets de large (le caractère cité plus haut s'inscrirait entre les octets 49152, 49153, 63488 et 63489).

	1ère colonne	2ème colonne			
1ère ligne	49152	49153	49154	19155	49156
	51200	51201	51202	51203	51204
	53248	53249	53250	53251	53252
	55296	55297	55298	55299	55300
	57344	57345	57346	57347	57348
	59392	59393	59394	59395	59396
	61440	61441	61442	61443	61444
	63488	63489	63490	63491	63492
2ème ligne	49232	49233	49234	49235	49236
	51280	51281	51282	51283	51284
	53328	53329	53330	53331	53332
	55376	55377	55378	55379	55380
	57424	57425	57426	57427	57428
	59472	59473	59474	59475	59476
	61520	61521	61522	61523	61524
	63568	63569	63570	63571	63572
⋮	⋮	⋮	⋮	⋮	⋮
24ème ligne	50992	50993	50994	50995	50996
	53040	53041	53042	53043	53044
	55088	55089	55090	55091	55092
	57136	57137	57138	57139	57140
	59184	59185	59186	59187	59188
	61232	61233	61234	61235	61236
	63280	63281	63282	63283	63284
	65328	65329	65330	65331	65332
25ème ligne	51072	51073	51074	51075	51076
	53120	53121	53122	53123	53124
	55168	55169	55170	55171	55172
	57216	57217	55218	55219	55220
	59264	59265	59266	59267	59268
	61312	61313	61314	61315	61316
	63360	63361	63362	63363	63364
	65408	65409	65410	65411	65412

		39ème colonne	40ème colonne		Numéro du trait
49227	49228	49229	49230	49231	1
51275	51276	51277	51278	51279	2
53323	53324	53325	53326	53327	3
55371	55372	55373	55374	55375	4
57419	57420	57421	57422	57423	5
59467	59468	59469	59470	59471	6
61515	61516	61517	61518	61519	7
63563	63564	63565	63566	63567	8
49307	49308	49309	49310	49311	1
51355	51356	51357	51358	51359	2
53403	53404	53405	53406	53407	3
55451	55452	55453	55454	55455	4
57499	57500	57501	57502	57503	5
59547	59548	59549	57550	59551	6
61595	61596	61597	61598	61599	7
63643	63644	63645	63646	63647	8
51067	51068	51069	51070	51071	1
53115	53116	53117	53118	53119	2
55163	55164	55165	55166	55167	3
57211	57212	57213	57214	57215	4
59259	59260	59261	59262	59263	5
61307	61308	61309	61310	61311	6
63355	63356	63357	63358	63359	7
65403	65404	65405	65406	65407	8
51147	51148	51149	51150	51151	1
53195	53196	53197	53198	53199	2
55243	55244	55245	55246	55247	3
57291	57292	57293	57294	57295	4
59339	59340	59341	59342	59343	5
61387	61388	61389	61390	61391	6
63435	63436	63437	63438	63439	7
65483	65484	65485	65486	65487	8

• En mode 2 enfin, chaque caractère s'inscrit sur une surface de la largeur de l'octet (le caractère cité plus haut s'inscrirait entre les octets 49152 et 63488).

Il faut parfois tenir compte de tout cela dans certains programmes, si l'on veut les utiliser en plusieurs modes.

4. QUELQUES ROUTINES DU SYSTÈME D'EXPLOITATION

ROUTINES "TEXTE"

&BB5D 47965
Représente un caractère sur l'écran, à la position actuelle du curseur de texte. Le curseur est ensuite décalé d'une position vers la droite.

Registre d'appel : le code du caractère doit être chargé dans A.

&BB5A 47962
Représente un caractère sur l'écran, ou *l'exécute s'il s'agit d'un caractère de commande* (le code 7, par exemple, produira un son de cloche — voir à ce sujet les pages 9.2, 9.3 et 9.4 du guide de l'utilisateur).
A doit être utilisé comme pour la routine précédente.

&BB75 47989
Fixe le curseur texte à une position donnée de l'écran. Le numéro de colonne doit être chargé dans H et le numéro de ligne dans L. Les routines &BB6F et &BB72 permettent respectivement de fixer le curseur texte à l'intérieur de la ligne où il se trouve, ou à une position donnée de la colonne où il se trouve. Les numéros de colonne ou de ligne doivent être chargés dans A.

&BB90 48016
Fixe la couleur de stylo. Le numéro de couleur doit être chargé dans A.

&BB96 48022
Fixe la couleur du papier. Le numéro de couleur doit être chargé dans A.

ROUTINES "GRAPHIQUE"

&BBC0 Fixe le curseur graphique à une position absolue. L'abscisse de cette position doit être chargée dans DE et l'ordonnée dans HL.

&BBC3 Fixe le curseur graphique à une position relative à la position actuelle du curseur. HL et DE doivent être employés comme précédemment.

&BBE1 Fixe la couleur de l'écriture graphique. Le numéro de couleur doit être chargé dans A.

&BBE4 Fixe la couleur du papier graphique. A doit être utilisé comme précédemment.

&BBEA Dessine un point à une position absolue. L'abscisse et l'ordonnée doivent respectivement être chargées dans DE et HL.

&BBED Dessine un point à une position relative au curseur graphique. DE et HL doivent être employés comme précédemment.

&BBF6 Dessine une droite à partir du curseur graphique jusqu'à une position absolue dont les coordonnées doivent être dans DE pour l'abscisse, et dans HL pour l'ordonnée.

&BBF9 Dessine une droite à partir du curseur graphique jusqu'à une position relative à ce curseur. HL et DE s'emploient comme précédemment.

&BBFC Écrit un caractère à la position actuelle du *curseur graphique*. Ce dernier détermine ~~l'angle~~ le coin supérieur gauche du caractère. Le curseur est ensuite déplacé d'une largeur de caractère. Cette largeur dépend bien entendu du mode. Le code du caractère doit être chargé dans A.

AUTRES ROUTINES

&BB06 Cette routine effectue une boucle tant qu'une touche n'est pas frappée. Lorsqu'une touche est frappée, le retour est effectué, et A contient le code du caractère correspondant à la touche. Signalons que, si la touche frappée est une touche de déplacement du curseur, A contient : F0 pour ←, F3 pour ↑, F1 pour → et F2 pour ↓. Cela est bien sûr particulièrement intéressant pour les jeux inter-actifs.

&BC14 Efface l'écran.

&BD19 Attend le retour du rayon. Cette routine permet la synchronisation des animations graphiques avec le balayage de l'écran.

ROUTINES "ARITHMÉTIQUE A VIRGULE FLOTTANTE"

&BD3D Copie une variable de (DE) dans (HL). DE doit être pointé sur l'adresse de la variable à recopier.

&BD40 Transforme une représentation entière en une représentation à virgule flottante. L'entier doit être chargé dans HL et l'adresse où doit être stockée la représentation en virgule flottante dans DE.

&BD46 Transforme une représentation en virgule flottante en un entier. HL doit être pointé sur l'adresse de la représentation en virgule flottante. Au retour, l'entier est dans HL.

&BD8B Calcule le cosinus d'un angle. HL doit être pointé sur l'adresse de la représentation en virgule flottante de l'angle. Le résultat est rangé à cette adresse.

&BD88 Même chose, mais pour le cosinus.

&BD5B Effectue l'opération (HL)–(DE). Le résultat est rangé dans (HL).

&BD5E Effectue l'opération (DE)–(HL). Le résultat est rangé dans (HL).

&BD58 Effectue l'opération (HL)+(DE). Le résultat est rangé dans (HL).

&BD64 Effectue l'opération (HL)/(DE). Le résultat est rangé dans (HL).

&BD6D Change le signe de (HL).

ROUTINES "ARITHMÉTIQUE ENTIÈRE"

&BDAC Effectue l'opération HL+DE. Le résultat est chargé dans HL.

&BDAF Effectue l'opération HL−DE. Le résultat est chargé dans HL.

&BDBE Effectue l'opération HL×DE. Le résultat est chargé dans HL.

&BDC1 Effectue l'opération HL/DE. Le résultat est chargé dans HL.

&BDC7 Inverse le signe de HL.

5. LES VARIABLES

LES VARIABLES NUMÉRIQUES

Comme vous le savez sans doute, le BASIC fait la différence entre ce que l'on appelle les nombres entiers et les nombres décimaux. Ces nombres ne sont pas traités de la même manière, et il existe d'ailleurs dans le système d'exploitation, deux blocs distincts de routines arithmétiques : l'un pour les variables entières et l'autre pour les variables à virgule flottante.

Il faut également savoir que le traitement des variables à virgule flottante est nettement plus long que celui des variables entières.

La ligne suivante, par exemple, sera exécutée en 5,6 secondes environ :

```
FOR I % =1 TO 10000 : NEXT I %
```

(Le symbole % signale à la machine qu'il s'agit d'un nombre entier.)
Alors que celle-ci le sera en plus de 10 secondes :

```
FOR I=1 TO 10000 : NEXT I
```

Nous allons nous intéresser à la manière dont ces variables sont stockées en mémoire par la machine.

Le cas des variables entières est simple : elles sont stockées sur deux cases mémoire. La première contient l'octet faible du nombre, et la seconde l'octet fort.

Nous pouvons le vérifier grâce au pointeur de variable que l'on peut appeler avec l'arobas (@). Cette fonction permet d'obtenir l'adresse d'une variable, c'est-à-dire l'adresse de l'emplacement mémoire contenant l'octet faible de cette variable (l'octet fort étant situé immédiatement après).

Essayez par exemple de taper ceci (en mode direct et après avoir réinitialisé la machine) :

```
A % = 4
```

Puis :

```
PRINT @ A %
```

Vous obtenez 374.

L'octet faible de A % est donc stocké dans l'emplacement mémoire 374 et l'octet fort dans l'emplacement 375. On peut le vérifier en faisant :

```
PRINT PEEK(374)
```

On obtient 4.

```
PRINT PEEK(375)
```

On obtient 0.

Un nombre peut être calculé à partir de son octet fort et de son octet faible grâce à la formule :

(octet fort * 256) + octet faible

(si l'on est sûr qu'il est positif, le problème étant un peu différent pour les nombres négatifs.)

Dans notre cas, nous avons bien : 4 = (0 * 256) + 4

Essayez maintenant ceci :

```
A % =500
```

Puis :

```
PRINT PEEK (@ A %) + PEEK (@ A % + 1) * 256
```

Vous obtenez bien 500.

Le premier PEEK donne le contenu de l'emplacement mémoire où est situé l'octet faible de A %, et le second le contenu de l'emplacement mémoire où est situé l'octet fort.

Pour conclure, retenons simplement qu'une variable entière est stockée sur 2 octets, et que l'octet faible est situé en "première position".

Le cas des variables à virgule flottante est un peu plus complexe, puisque 5 octets sont nécessaires pour leur stockage en mémoire (pour information, les quatre premiers octets s'appellent la *mantisse*, et le cinquième l'*exposant*).

Pour ces variables à virgule flottante, la fonction @ donnera l'adresse du premier des cinq emplacements mémoire où elle est stockée.

Notons enfin qu'il est possible de transmettre l'adresse d'une variable grâce à l'arobas lors de l'utilisation de l'instruction CALL :

CALL Adresse du programme, @ variable

Nous utilisons cette possibilité dans le dernier programme de cet ouvrage (Tri de données alphanumériques).

Vous en savez maintenant assez sur les variables numériques pour étudier les programmes présentés.

LES VARIABLES ALPHANUMÉRIQUES

Les variables alphanumériques sont des variables pouvant être constituées de lettres, de chiffres, ou des deux ensembles. Elles sont signalées par le symbole $. Par exemple :

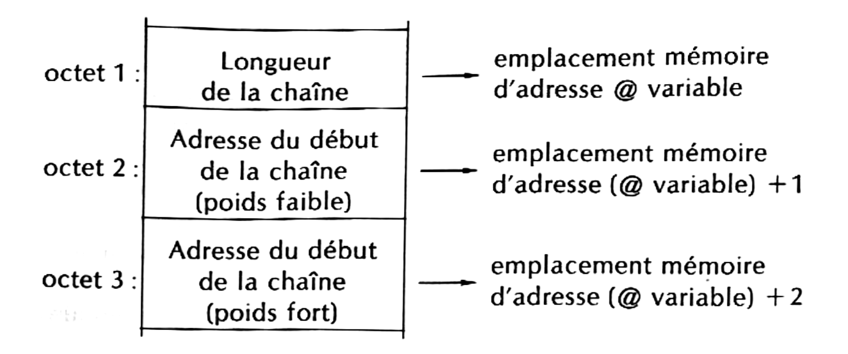
A$ = "JAMES BOND 007"

Une chaîne alphanumérique (c'est-à-dire comprenant plusieurs éléments alphanumériques) est bien sûr stockée en mémoire sous la forme d'une succession des codes ASCII des éléments qui la composent. La chaîne "JAMES BOND 007", par exemple, se présentera ainsi :

&4A (code ASCII de J)
&41 (code ASCII de A)
&4D (code ASCII de M)
&45 (code ASCII de E)
&53 (code ASCII de S)
&20 (code ASCII de l'espace)
etc.

Là encore, l'arobas peut être utilisé.

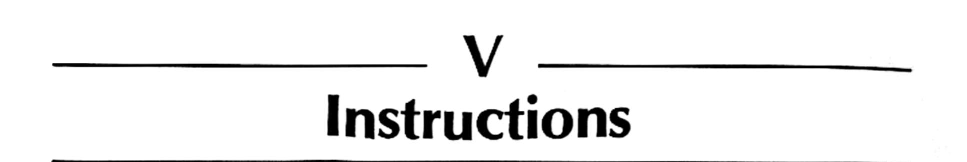
Dans ce cas, pourtant, l'adresse obtenue ne sera pas directement l'adresse de la variable, mais l'adresse de ce que l'on appelle son *descripteur de chaîne*. Ce descripteur comporte 3 octets :



Notons que la longueur d'une chaîne de caractères (ou, si l'on veut, son nombre d'éléments) ne pouvant dépasser 255, le chiffre indiquant cette longueur peut toujours tenir sur 1 octet.

```
EXEMPLE:

1 A$="JAMES BOND":L=PEEK(@A$):ADR=PEEK(@
A$+1)+PEEK(@A$+2)*256:FOR I=0 TO L-1:PRI
NT CHR$(PEEK(ADR+I));:NEXT
RUN
JAMES BOND
```



V

Instructions

Cette partie récapitule et décrit d'une manière formelle les instructions utilisées dans les programmes présentés.

Celles-ci sont classées dans l'ordre alphabétique de leurs mnémoniques, dont les représentations conventionnelles notables sont les suivantes :

- Les registres doubles sont, selon le cas, symbolisés par ss, rr ou dd.
- Les registres simples sont symbolisés par r.
- Les données immédiates tenant sur un octet (ou si l'on veut, les valeurs quelconques n'ayant qu'un octet faible) sont représentées par n. Les autres sont représentées par nn.
- L'adresse d'un emplacement mémoire est représentée par nn, et le contenu de cet emplacement mémoire par (nn).
- La partie basse d'un registre double est celle où se trouvent les octets de poids faible. La partie haute est celle où se trouvent les octets de poids fort.

Signalons d'autre part que :

- Une description détaillée de l'instruction est proposée lorsque cela semble nécessaire.
- Seuls les éventuels effets sur les indicateurs S et Z ont été spécifiés.
- Concernant toutes les instructions de chargement, la règle générale est que la source n'est pas modifiée par l'instruction (cela concerne également les instruction de transfert répétitif).

ADD HL,ss (Additionner HL et le registre double ss)

Octet 1 : Selon registre ss (BC:09 – DE:19 – HL:29)

Description : Le contenu du registre double spécifié est additionné au contenu de HL. Le résultat est rangé dans HL, l'autre registre restant inchangé.

Indicateurs : aucun effet.

BIT b,r (Test du bit b du registre r)

Octet 1 : CB
Octet 2 : Selon le registre et le bit

Numéro de bit	A	B	C	D	E	H	L
0	47	40	41	42	43	44	45
1	4F	48	49	4A	4B	4C	4D
2	57	50	51	52	53	54	55
3	5F	58	59	5A	5B	5C	5D
4	67	60	61	62	63	64	65
5	6F	68	69	6A	6B	6C	6D
6	77	70	71	72	73	74	75
7	7F	78	79	7A	7B	7C	7D

Indicateurs : L'indicateur Z est mis à 1 si le bit testé est 0, et il est mis à 0 autrement. L'indicateur S est modifié aléatoirement.

CALL pq (Appel du sous-programme d'adresse pq)

Octet 1 : CD

Octet 2 : adresse, poids faible

Octet 3 : adresse, poids fort

Observation ; Lorsque cette instruction est rencontrée, l'adresse de retour du sous-programme est déposée sur la pile.

Indicateur : aucun effet.

CP s (Comparaison de l'opérande s et de l'accumulateur A)

Si s est un registre :
Octet 1 : selon registre (A:BF – B:B8 – C:B9 – D:BA – E:BB – H:BC – L:BD)

Si s est une donnée immédiate :
Octet 1 : FE
Octet 2 : donnée immédiate

Si s est (HL) :
Octet 1 : BE

Si s est (IX+d) :
Octet 1 : DD
Octet 2 : BE
Octet 3 : d

Indicateurs : L'indicateur Z est mis à 1 quand le test indique l'égalité. Pour effectuer la comparaison, le processeur soustrait l'opérande de l'accumulateur, mais sans conserver le résultat (l'accumulateur reste inchangé). L'indicateur S est par conséquent modifié selon le résultat.

DEC m (Décrémentation de l'opérande m)

Si r est un registre → Octet 1 : selon registre (A:3D – B:05 – C:0D – D:15 – E:1D – H:25 – L:2D)

Description : Le contenu du registre spécifié est diminué de 1.

Indicateurs : S et Z sont modifiés selon le résultat.

DEC rr (Décrémentation du registre double rr)

Octet 1 : selon registre (BC:0B – DE:1B – HL:2B)

Indicateurs : aucun effet.

DEC IX (Décrémentation du registre IX)

Octet 1 : DD
Octet 2 : 2B

Indicateurs : aucun effet.

DJNZ e (Décrémentation de B et saut relatif de e si non nul)

Octet 1 : 10
Octet 2 : longueur du saut

Indicateurs : aucun effet

EX DE, HL (Échange des registres HL et DE)

Octet 1 : EB

Indicateurs : aucun effet.

INC rr (Incrémentation du registre double rr)

Octet 1 : selon registre (BC:03 – DE:13 – HL:23)

Description : Le contenu du registre spécifié est augmenté de 1.

Indicateurs : aucun effet.

INC IX (Incrémentation du registre IX)

Octet 1 : DD
Octet 2 : 23

Indicateurs : aucun effet.

JP cc, pq (Saut conditionnel à l'adresse pq)

Octet 1 : selon condition (positif : F2 ; négatif : FA ; nul : CA ; non nul : C2)
Octet 2 : adresse, poids faible
Octet 3 : adresse, poids fort

Description : Si la condition spécifiée est remplie, le programme saute à l'adresse indiquée. Sinon, cette adresse est ignorée et le programme se poursuit normalement.

Indicateurs : aucun effet.

JP pq (Saut à l'adresse pq)

Octet 1 : C3
Octet 2 : adresse, poids faible
Octet 3 : adresse, poids fort

Description : Le programme saute à l'adresse indiquée.

Indicateurs : aucun effet.

JR cc, e (Saut relatif conditionnel de e)

Octet 1 : selon condition (nul : 28 – non nul : 20)
Octet 2 : longueur du saut

Indicateurs : aucun effet.

JR e (Saut relatif de e)

Octet 1 : 18
Octet 2 : longueur du saut

Indicateurs : aucun effet.

LD dd, (nn) (Chargement du registre double dd à partir de l'emplacement mémoire d'adresse nn)

Octet 1 : ED

Octet 2 : selon registre (BC:4B – DE:5B – HL:6B)

Octet 3 : adresse, poids faible

Octet 4 : adresse, poids fort

Description : Le contenu de l'emplacement mémoire dont l'adresse est spécifiée est chargé dans les poids faibles du registre choisi. Le contenu de l'emplacement mémoire suivant immédiatement le précédent est chargé dans les poids fort du registre.

Indicateurs : aucun effet.

LD dd, nn (chargement du registre double dd avec la donnée immédiate nn)

Octet 1 : selon registre (BC:1 – DE:11 – HL:21)

Octet 2 : donnée immédiate, poids faible

Octet 3 : donnée immédiate, poids fort

Indicateurs : aucun effet.

LD r, n (Chargement du registre r avec la donnée immédiate n)

Octet 1 : selon registre (A:3E – B:06 – C:0E – D:16 – E:1E – H:26 – L:2E)

Octet 2 : donnees immédiate

Indicateurs : aucun effet.

LD r, r' (Chargement du registre r à partir du registre r')

Octet 1 : selon registre

destinataire \ (source)	A	B	C	D	E	H	L
A	7F	78	79	7A	7B	7C	7D
B	47	40	41	42	43	44	45
C	4F	48	49	4A	4B	4C	4D
D	57	50	51	52	53	54	55
E	5F	58	59	5A	5B	5C	5D
H	67	60	61	62	63	64	65
L	6F	68	69	6A	6B	6C	6D

Indicateurs : aucun effet.

LD (BC), A (Chargement de l'emplacement mémoire adressé par BC à partir de l'accumulateur)

Octet 1 : 2

Indicateurs : aucun effet.

LD (DE), A (Chargement de l'emplacement mémoire adressé par DE à partir de l'accumulateur)

Octet 1 : 12

Indicateurs : aucun effet.

LD (HL), n (Chargement de la donnée immédiate n dans l'emplacement mémoire adressé par HL)

Octet 1 : 36
Octet 2 : donnée immédiate

Description : La donnée fournie est chargée dans l'emplacement mémoire dont l'adresse est dans HL.

Indicateurs : aucun effet.

LD r, (IX+d) (Chargement du registre r à partir de l'emplacement mémoire d'adresse IX+d)

Octet 1 : DD
Octet 2 : selon registre (A:7E – B:46 – C:4E – D:56 – E:5E – H:66 – L:6E)
Octet 3 : déplacement

Description : Le contenu de l'emplacement mémoire dont l'adresse est IX+d (d est appelé "déplacement"), est chargé dans le registre spécifié.

Indicateurs : aucun effet.

LD (nn), dd (Chargement de l'emplacement mémoire d'adresse nn à partir du registre double dd)

Octet 1 : ED
Octet 2 : selon registre (BC:43 – DE:53 – HL:63)
Octet 3 : adresse, poids faible
Octet 4 : adresse, poids fort

Description : Le contenu de la partie basse du registre spécifié (c'est-à-dire la partie contenant l'octet faible) est chargé dans l'emplacement mémoire indiqué. Le contenu de la partie haute du registre est chargé dans l'emplacement mémoire suivant immédiatement le précédent.

Indicateurs : aucun effet.

LD (nn), HL (Chargement de l'emplacement mémoire d'adresse nn à partir de HL)

Octet 1 : 22
Octet 2 : adresse, poids faible
Octet 3 : adresse, poids fort

Description : Le contenu du registre L est chargé dans l'emplacement mémoire dont l'adresse est spécifiée. Le contenu du registre H est chargé dans l'emplacement mémoire suivant immédiatement le précédent.

Indicateurs : aucun effet.

LD A, (BC) (Chargement de l'accumulateur à partir de l'emplacement mémoire adressé par BC)

Octet 1 : A

Indicateurs : aucun effet.

LD A, (DE) (Chargement de l'accumulateur à partir de l'emplacement mémoire adressé par DE)

Octet 1 : 1A

Indicateurs : aucun effet.

LD HL, (nn) (Chargement de HL à partir de l'emplacement mémoire d'adresse nn)

Octet 1 : 2A
Octet 2 : adresse, poids faible
Octet 3 : adresse, poids fort

Description : Le contenu de l'emplacement mémoire dont l'adresse est spécifiée est chargé dans le registre L. Le contenu de l'emplace-

ment mémoire suivant immédiatement le précédent est chargé dans H.

Indicateurs : aucun effet.

LD (HL), r (Chargement de l'emplacement mémoire adressé par HL à partir du registre r)

Octet 1 : selon registre (A:77 – B:70 – C:71 – D:72 – E:73 – H:74 – L:75)

Indicateurs : aucun effet.

LD (IX+d), r (Chargement de l'emplacement mémoire d'adresse IX+d à partir du registre r)

Octet 1 : DD

Octet 2 : selon registre (A:77 – B:70 – C:71 – D:72 – E:73 – H:74 – L:75)

Octet 3 : d

Indicateurs : aucun effet.

LD IX, (nn) (Chargement du registre IX à partir de l'emplacement mémoire d'adresse nn)

Octet 1 : DD

Octet 2 : 2A

Octet 3 : adresse, poids faible

Octet 4 : adresse, poids fort

Description : Le contenu de l'emplacement mémoire dont l'adresse est spécifiée est chargé dans la partie basse du registre IX. Le contenu de l'emplacement mémoire suivant immédiatement le précédent est chargé dans la partie haute.

Indicateurs : aucun effet.

LDDR (Transfert répétitif de bloc avec décrémentation)

Octet 1 : ED

Octet 2 : B8

Description : Le contenu de l'emplacement mémoire adressé par HL est chargé dans l'emplacement mémoire adressé par DE. Les trois registres HL, DE et BC sont ensuite décrémentés. Si BC est

différent de 0, l'instruction est exécutée de nouveau. Sinon, le programme se poursuit en séquence.

Indicateurs : aucun effet.

LDIR (Transfert répétitif de bloc avec incrémentation)

Octet 1 : ED

Octet 2 : B0

Description : Le contenu de l'emplacement mémoire adressé par HL est chargé dans l'emplacement mémoire adressé par DE. Les registres HL et DE sont ensuite incrémentés, alors que BC est décrémenté. Si BC est différent de 0, l'instruction est exécutée de nouveau. Sinon le programme se poursuit en séquence.

Indicateurs : aucun effet.

POP qq (Dépilement du registre double qq)

Octet 1 : selon registre (BC:C1 – DE:D1 – HL:E1 – AF:F1)

Description : La donnée située au-dessus de la pile est enlevée et chargée dans le registre spécifié.

Indicateurs : aucun effet.

PUSH qq (Empilement du registre double qq)

Octet 1 : selon registre (BC:C5 – DE:D5 – HL:E5 – AF:F5)

Description : Le contenu du registre spécifié est déposé sur la pile (le registre reste malgré tout inchangé)

Indicateurs : aucun effet.

RET (Retour de sous-programme)

Octet 1 : C9

Observation : Lorsque cette instruction est rencontrée, l'adresse de retour du sous-programme est retirée de la pile.

Indicateurs : aucun effet.

RET cc (Retour conditionnel de sous-programme)

Octet 1 : selon condition (non nul : C0 – nul : C8 – positif : F0 – négatif : F8)

Observation : Dans ce cas particulier, l'adresse de retour n'est retirée de la pile que si ce retour a effectivement lieu, donc si la condition spécifiée est remplie.

Indicateurs : aucun effet.

SBC HL, ss (Soustraire de HL le registre double ss)

Octet 1 : ED

Octet 2 : selon registre (BC:42 – DE:52 – HL:62)

Description : Le contenu du registre double spécifié est soustrait du contenu du registre HL et le résultat est rangé dans HL (l'autre registre reste inchangé).

Indicateurs : S est Z sont modifiés selon le résultat.

CONCLUSION

Nous espérons, naturellement, que cet ouvrage aura répondu à votre attente, et que vous aurez éprouvé plus de plaisir que de peine à en suivre les méandres.

Pour notre part, le but que nous nous étions fixé serait atteint si, après que vous ayez tourné la dernière page, vous vous rendiez compte que le "virus" vous a contaminé, et que vous décidiez de vous mettre à faire vos propres programmes en langage machine.

N'essayez pas, bien sûr, de vous attaquer immédiatement à des réalisations trop ambitieuses (nous n'avons pas non plus appris le langage machine en deux jours), mais rappelez-vous qu'après un minimum de pratique, vous ne serez plus limité que par votre propre imagination.

A ceux qui, d'ores et déjà, se sentiraient "mordus", nous donnerons deux ultimes conseils :

- N'oubliez pas, d'abord, qu'une partie seulement des instructions du Z 80 a été utilisée dans nos programmes. Il en existe beaucoup d'autres, certes parfois plus complexes, mais toutes aussi excitantes et susceptibles de vous ouvrir des perspectives nouvelles. La première chose à faire est donc sans doute d'acheter un ouvrage traitant ce sujet d'une manière exhaustive. Il deviendra vite, soyez-en sûr, votre bible de référence.

- Sachez également que vous trouverez grand profit à utiliser un programme assembleur (on en trouve pour l'Amstrad à des prix dérisoires). Sans entrer dans les détails, il suffit de savoir que l'assembleur permet une manipulation beaucoup plus facile des instructions. Par exemple, il utilise les mnémoniques au lieu des codes machine ; il délivre le cas échéant des messages d'erreur au lieu de faire "tout sauter", ainsi que des listings compréhensibles et avec commentaires ; et surtout il permet ce que l'on appelle des *break points,* c'est-à-dire une vérification de l'état des registres et de la pile à tel ou tel moment précis du programme.

Quoiqu'il en soit, nous vous souhaitons bon courage et bons programmes pour vos confrontations futures avec ce langage qui, outre qu'il offre des possibilités extraordinaires, qui ne pourraient même pas être envisagées en BASIC, est bien le seul capable de procurer l'exaltante sensation de plonger véritablement dans les entrailles de la machine.

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par Rodnay Zaks,*
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par Rodnay Zaks,*
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par Rodnay Zaks,*
636 pages, Réf. 0040

TECHNIQUES D'INTERFACE aux microprocesseurs *par Austin Lesea et Rodnay Zaks,*
450 pages, Réf. 0039

LEXIQUE INTERNATIONAL MICRO-ORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par Joël Poncet,*
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par Pierre Le Beux,*
140 pages, Réf. 369

LA SOLUTION RS-232 *par Joe Campbell,*
208 pages, Réf. 0052

MINITEL ET MICRO-ORDINATEUR *par Pierrick Bourgault,*
198 pages, Réf. 0119

ROBOTS - CONSTRUCTION, PROGRAMMATION *par Fernand Esteves,*
400 pages, Réf. 0130

ALGORITHMES *par P. Beaufils et W. Luther,*
296 pages, Réf. 0149

BASIC

VOTRE PREMIER PROGRAMME BASIC *par Rodnay Zaks,*
208 pages, Réf. 263

INTRODUCTION AU BASIC *par Pierre Le Beux,*
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices *par Jean-Pierre Lamoitier,*
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par Xuan Tung Bui,*
204 pages, Réf. 253

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par Alan R. Miller,*
318 pages, Réf. 259

BASIC, PROGRAMMATION STRUCTURÉE *par Richard Mateosian,*
352 pages, Réf. 0129

JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
204 pages, Réf. 247

FICHIERS EN BASIC *par Alan Simpson,*
256 pages, Réf. 0102

TECHNIQUES DE PROGRAMMATION EN BASIC *par S. Crosmarie, M. Perron et D. Philippine*
152 pages, Réf. 0124

PASCAL

INTRODUCTION AU PASCAL *par Pierre Le Beux,*
496 pages, Réf. 0030

LE PASCAL PAR LA PRATIQUE *par Pierre Le Beux et Henri Tavernier,*
562 pages, Réf. 361

LE GUIDE DU PASCAL *par Jacques Tiberghien,*
504 pages, Réf. 423

PROGRAMMES EN PASCAL pour Scientifiques e Ingénieurs *par Alan R. Miller,*
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux,*
366 pages, Réf. 360

MICRO-ORDINATEURS

ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut,*
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 376

ALICE, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 401

ALICE, GUIDE DE L'UTILISATEUR *par Norbert Rimoux,*
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
192 pages, Réf. 420

AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par Rodnay Zaks*
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0108

AMSTRAD, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
208 pages, Réf. 0136

AMSTRAD EXPLORÉ *par John Braga,*
192 pages, Réf. 0135

AMSTRAD, GUIDE DU GRAPHISME *par James Wynford*
208 pages, Réf. 0141

AMSTRAD CP/M 2.2 *par Anatole D'Hardencourt,*
248 pages, Réf. 0156

AMSTRAD ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
160 pages, Réf. 0167

AMSTRAD MULTIPLAN *de Microsoft,*
496 pages, Réf. 1111

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II,
Tomes 1 et 2 *par Léopold Laurent,*
208 pages, Réf. 333 et 380

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE
par Douglas Hergert et Joseph T. Kalash,
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert,*
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
par Joseph Caggiano,
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
par Thomas Blackadar,
160 pages, Réf. 0089

MULTIPLAN SUR MACINTOSH
par Goulven Habasque,
240 pages, Réf. 0099

INTRODUCTION A MAC PASCAL *par Pierre Le Beux,*
416 pages, Réf. 0145

MACINTOSH POUR LA PRESSE, L'EDITION ET
LA PUBLICITE *par Bernard Le Du,*
160 pages, Réf. 0173

ATARI

JEUX EN BASIC SUR ATARI *par Paul Bunn,*
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par Thomas Blackadar,*
192 pages, Réf. 354

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut,*
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost,*
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
par Pierre Monsaut,
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert,
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
par J. Kascmer,
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
par Stanley R. Trost,
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
par Charles Platt,
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par Eric Ravis,*
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
par Marty Dejonghe et Caroline Earhart,
208 pages, Réf. 390

COMMODORE 64, BASIC APPROFONDI
par Gary Lippman,
216 pages, Réf. 0100

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut,*
96 pages, Réf. 324

EXL 100

EXL 100, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0126

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
par François Abella,
208 pages, Réf. 264

HECTOR

HECTOR JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 388

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier,*
256 pages, Réf. 338

IBM PC GUIDE DE L'UTILISATEUR
par Joan Lasselle et Carol Ramsey,
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 359

GRAPHIQUES SUR IBM PC *par Nelson Ford,*
320 pages, Réf. 357

GUIDE DE PC DOS *par Richard A. King,*
240 pages, Réf. 0013

LASER

LASER JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 371

MO 5

MO 5 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 370

MO 5, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 375

MO 5, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 384

MO 5, DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par Daniel Lebigre,*
272 pages, Réf. 0118

MO 5, STATIQUE, DYNAMIQUE, ELECTRONIQUE, PROGRAMMES DE PHYSIQUE EN BASIC
par Beaufils, Lamarche et Muggianu,
240 pages, Réf. 0148

MO 5, PROGRAMMES D'ELECTRONIQUE EN BASIC
par Beaufils, Delusurieux, Do, Romanacce,
312 pages, Réf. 0143

MO 5, OPTIQUE, THERMODYNAMIQUE, CHIMIE
par P. Beaufils, M. Lamarche, Y. Muggianu,
224 pages, Réf. 0161

MSX

MSX, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks,*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0109

MSX, GUIDE DU GRAPHISME *par Mike Shaw,*
192 pages, Réf. 0132

MSX, PROGRAMMES EN LANGAGE MACHINE
par Steeve Webb,
112 pages, Réf. 0153

MSX, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
216 pages, Réf. 0144

MSX, GUIDE DU BASIC *par Michel Laurent,*
264 pages, Réf. 0155

MSX, JEUX EN ASSEMBLEUR *par Eric Ravis*
112 pages, Réf. 0170

MSX, ROUTINES GRAPHIQUES EN ASSEMBLEUR
par Steeve Webb
88 pages, Réf. 0154

MSX, TECHNIQUES DE PROGRAMMATION DES JEUX EN ASSEMBLEUR
par Georges Fagot-Barraly,
176 pages, Réf. 0178

MSX ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
157 pages, Réf. 0168

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir,
2 tomes, Réf. 261-262

SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsau*
96 pages, Réf. 377

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
par S.M. Gee,
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Za*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
par François Abella,
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 350

JEUX SUR TO 7 et MO 5 *par Georges Fagot-Barraly,*
168 pages, Réf. 0134

GESTION DE FICHIERS SUR TO 7 ET MO 5
par Jean-Pierre Lhoir,
136 pages, Réf. 0127

TO 7, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 374

TO 7 / MO 5, GUIDE DU BASIC
par Jean-Louis Greco et Michel Laurent,
288 pages, Réf. 0158

TO 7 / MO 5 ASTROLOGIE/NUMEROLOGIE/BIORYTHME
par Pierrick Bourgault,
160 pages, Réf. 0169

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
par Léopold Laurent,
2 tomes, Réf. 366-251

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsau*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
par Pierre Monsaut,
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
par Orson Kellog,
112 pages, Réf. 300

TRS-80 COULEUR, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 414

TRS-80 COULEUR, 56 PROGRAMMES
par Stanley R. Trost,
160 pages, Réf. 413

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
par G. O. Hamann,
2 tomes, Réf. 329-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0128

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 351

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 0058

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 0181

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

PROGRAMMATION DU 6800
par Daniel-Jean David et Rodnay Zaks,
374 pages, Réf. 327

PROGRAMMATION DU 6809
par Rodnay Zaks et William Labiak,
392 pages, Réf. 0139

PROGRAMMATION DU 8086/8088
par James W. Coffron,
304 pages, Réf. 0016

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 0133

ASSEMBLEUR DU 8086/8088 *par François Retoreau,*
616 pages, Réf. 0093

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD
par Charles W. Grant et Jon Butah,
308 pages, Réf. 365

GUIDE DE MS-DOS *par Richard A. King,*
360 pages, Réf. 0117

INTRODUCTION A UNIX *par John D. Halamka,*
240 pages, Réf. 0098

GUIDE DE PRODOS
par Pierre Beaufils et Wolfram Luther,
248 pages, Réf. 0146

APPLICATIONS ET LOGICIELS

INTRODUCTION AU TRAITEMENT DE TEXTE
par Hal Glatzer,
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 0062

WORDSTAR APPLICATIONS *par Julie Anne Arca,*
320 pages, Réf. 0005

VISICALC APPLICATIONS *par Stanley R. Trost,*
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par Dominique Helle,*
304 pages, Réf. 309

INTRODUCTION A dBASE II *par Alan Simpson,*
280 pages, Réf. 0064

DE VISICALC A VISI ON *par Jacques Bourdeu,*
256 pages, Réf. 321

MULTIPLAN POUR L'ENTREPRISE
par D. Helle et G. Boussand,
304 pages, Réf. 0079

dBASE II APPLICATIONS *par Christophe Stehly,*
248 pages, Réf. 416

INTRODUCTION A LOTUS 1-2-3
par Chris Gilbert et Laurie Williams,
272 pages, Réf. 0106

INTRODUCTION A dBASE III *par Alan Simpson,*
272 pages, Réf. 0131

LOTUS 1-2-3 POUR L'ENTREPRISE
par Dominique Helle et Guy Boussand,
256 pages, Réf. 0147

LOTUS 1-2-3 PROGRAMMATION DES MACRO-COMMANDES *par Goulven Habasque,*
144 pages, Réf. 0150 F

LOGISTAT, ANALYSE STATISTIQUE DES DONNÉES
par Fredj Tekaia et Michele Bidel,
352 pages, Réf. 0115

ALGORITHMES *par P. Beaufils, et W. Luther,*
296 pages, Réf. 0149

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 42.03.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 61 80 2-0
Telex : 08588163



Paris • Berkeley • Düsseldorf

Achevé d'imprimer le 13 janvier 1986 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 1er trimestre 1986 - Imprimeur n° 1937

L'Amstrad CPC 464 est équipé d'un processeur Z80, fonctionnant à 4 mégahertz et disposant d'un jeu d'instructions extrêmement puissant.
En outre, la qualité de son BASIC, tant du point de vue de la vitesse que du nombre d'instructions, range cette machine parmi les plus performantes du marché.
Il n'en reste pas moins que le BASIC sera toujours le BASIC, avec ses inconvénients et ses limites. Qui, par exemple, n'a pas éprouvé d'amères déceptions en voyant un mobile se traîner sur l'écran, avec force soubresauts, au terme d'un programme d'animation en BASIC ?
De plus, on ne peut que regretter l'absence, sur l'Amstrad d'instructions telles que PAINT, CIRCLE, etc.
Cet ouvrage contient de nombreux exemples de programmes :
— Pause de X secondes.
— Intégration d'un programme machine.
— Dessin d'un quadrilatère.
— Défilement d'une ligne.
— Dessin d'un cercle.
— Cercle rapide.
— Déplacement d'un mobile.
— Tri de données alphanumériques.
— Ainsi qu'une bibliothèque d'utilitaires.




AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
AC
ME
https://acpc.me/